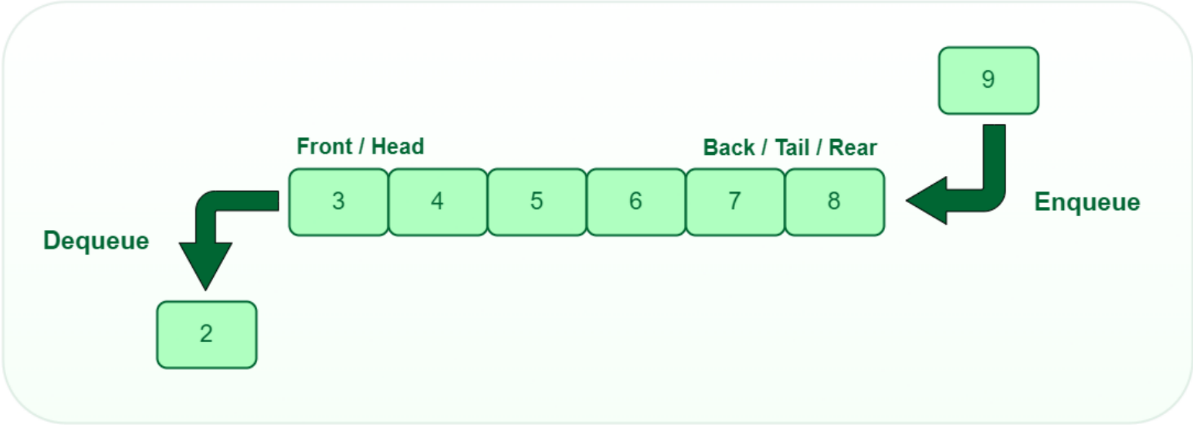
|  |  |
|---|---|
| **Name** | **Virinchi Sadashiv Shettigar** |
| **UID no.** | **2021300118** |
| **Experiment No.** | 2 |

|  |  |
|---|---|
| **AIM:** | Que Operations Menu driven program |

## Program 1

**THEORY:**

**Queue:** Queue is a linear data structure. It works in a FIRST IN FIRST OUT (FIFO) manner. We can perform operations on queues from both sides. We define a queue to be a list in which all additions to the list are made at one end, and all deletions from the list are made at the other end. The element which is first pushed into the order, the operation is first performed on that.



**Operations associated with queues:**

1. Enqueue():
   - Queues maintain two data pointers, front and rear. Therefore, its operations are comparatively difficult to implement than that of stacks.
   - The following steps should be taken to enqueue (insert) data into a queue −
   - Step 1 − Check if the queue is full.
   - Step 2 − If the queue is full, produce overflow error and exit.
   - Step 3 − If the queue is not full, increment rear pointer to point the next empty space.
   - Step 4 − Add data element to the queue location, where the rear is pointing.
   - Step 5 − return success.

2. Dequeue():
    - Accessing data from the queue is a process of two tasks − access the data where front is pointing and remove the data after access. The following steps are taken to perform dequeue operation −
    -
    - Step 1 − Check if the queue is empty.
    -
    - Step 2 − If the queue is empty, produce underflow error and exit.
    -
    - Step 3 − If the queue is not empty, access the data where front is pointing.
    -
    - Step 4 − Increment front pointer to point to the next available data element.
    -
    - Step 5 − Return success.
3. Peek(): Gets the element at the front of the queue without removing it.
4. isfull():  Checks if the queue is full
5. isempty():  Checks if the queue is empty.

**Applications of Queue**

1. CPU scheduling, Disk Scheduling

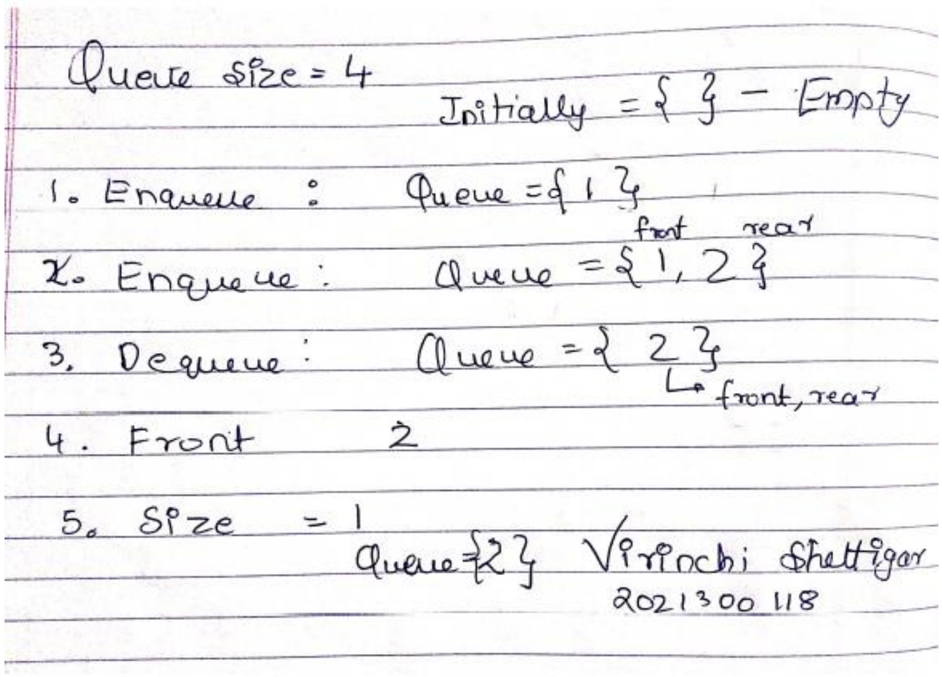
2. When data is transferred asynchronously between two processes.The queue is used for synchronization. For example: IO Buffers, pipes, file IO, etc.

3. Call Center phone systems use Queues to hold people calling them in order.

4. Handling of interrupts in real-time systems.

**ALGORITHM:**

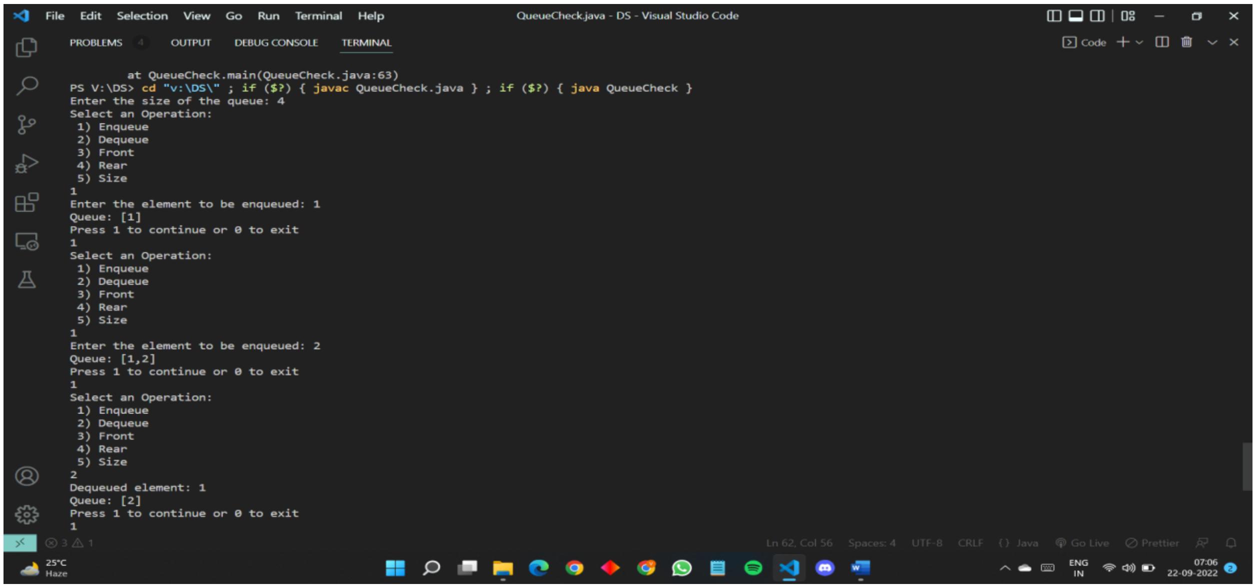
```
class Queue
    Main method:
        1. Input an integer size from users
        2. Pass the value of integer in the object class of Queue.
        3. Loop switch case until user exit from the code
        4. Using Switch case performing various operation such as Enqueue , Dequeue,
           Front,Rear, Size,etc.

class intQueue:
  int front
  int rear
```

| | |
|---|---|
| | int size<br>intQueue method:<br>1. Initialize the array size<br>2. Front=0 and rear=-1(initially)<br>3. Initialize the size<br>Enqueue method:<br>1. If the queue is not full then increment the rear by 1 and insert the element at the back<br>2. Else Queue Full<br>Dequeue method:<br>1. If the queue is not empty then increment front by 1 and delete element<br>2. Else Queue Empty<br>Front method:<br>1. If queue is not empty then print the front element of the queue.<br>2. Else Queue Empty<br>Front method:<br>1. If queue is not empty then print the rear element of the queue.<br>2. Else Queue Empty<br>Print method:<br>1. Iterate through every element of queue from front to rear and print them<br>isEmpty method:<br>1. Return true if front>rear or else return false<br>isFull method:<br>1. Return true if rear=size-1 or else return false |
| **PROBLEM-SOLVING:** | <br>Queue size = 4<br>Initially = { } - Empty<br>1. Enqueue : Queue = { 1 }<br>front rear<br>2. Enqueue : Queue = { 1, 2 }<br>3. Dequeue : Queue = { 2 }<br>↳ front, rear<br>4. Front 2<br>5. Size = 1<br>Queue={2} Virinchi Shettigar<br>2021300118 |

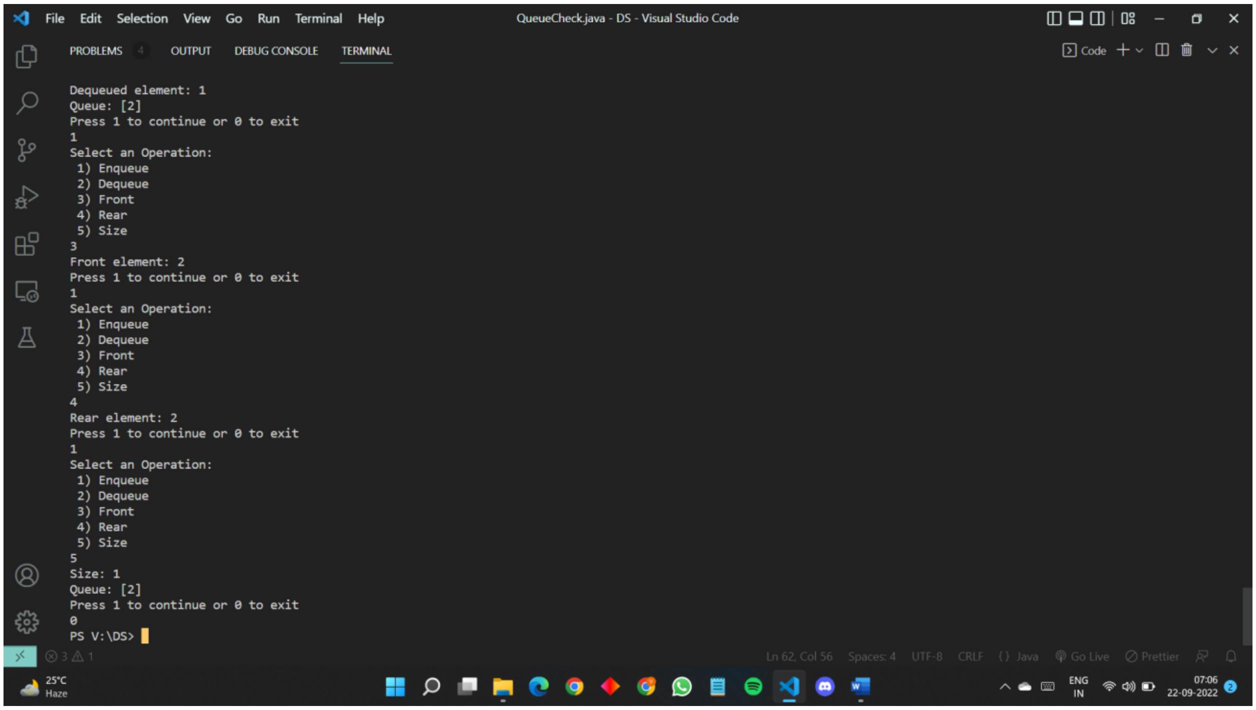
**PROGRAM:**

```
   import java.util.Scanner;
class IntQueue {
    int[] queue;
    int front;
    int rear;
    int capacity;
    public IntQueue(int size) {
        queue = new int[size];
        capacity = size;
        front = 0;
        rear = -1;
    }
    public void enqueue(int e){
        if(isFullQueue()) {
            System.out.println("Queue is full!");
        }
        queue[++rear] = e;
    }
    public int dequeue() {
        if(isEmptyQueue()) {
            System.out.println("Queue is empty!");
        }
        return queue[front++];
    }
    public int front() {
        if(isEmptyQueue()) {
            System.out.println("Queue is empty!");
        }
        return queue[front];
    }
    public int rear() {
        if(isEmptyQueue()) {
            System.out.println("Queue is empty!");
        }
        return queue[rear];
    }
    public String printQueue() {
        String s = "[";
        for(int i=front;i<rear+1;i++) {
            s += queue[i]+(i!=rear?",":"");
        }
        return s += "]";
    }
    public boolean isEmptyQueue() {
        return front>rear;
    }
    public boolean isFullQueue() {
        return rear == capacity - 1;
    }
    public int size() {
        return rear+1-front;
    }
}
class Queue {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter the size of the queue: ");
        int n = sc.nextInt();
        IntQueue queue = new IntQueue(n);
        int flag,choice;
        while(true) {
            System.out.println("Select an Operation:\n 1) Enqueue\n 2) Dequeue\n 3) Front\n 4) Rear \n 5) Size");
            choice = sc.nextInt();
            switch(choice) {
                case 1:
                    System.out.print("Enter the element to be enqueued: ");
                    int e = sc.nextInt();
                    try {
                        queue.enqueue(e);
                        System.out.println("Queue: " + queue.printQueue());
                    } catch(Exception ex) {
                        System.out.println(ex.getMessage());
                    }
                    break;
                case 2:
                    try {
                        System.out.println("Dequeued element: " + queue.dequeue());
                        System.out.println("Queue: " + queue.printQueue());
                    } catch(Exception ex) {
                        System.out.println(ex.getMessage());
                    }
                    break;
                case 3:
                    try {
                        System.out.println("Front element: " + queue.front());
                    } catch(Exception ex) {
                        System.out.println(ex.getMessage());
                    }
                    break;
                case 4:
                    try {
                        System.out.println("Rear element: " + queue.rear());
                    } catch(Exception ex) {
                        System.out.println(ex.getMessage());
                    }
                    break;
                case 5:
                    System.out.println("Size: " + queue.size());
                    System.out.println("Queue: " + queue.printQueue());
                    break;
                default:
                    System.out.println("Invalid choice!");
            }
            System.out.println("Press 1 to continue or 0 to exit");
            flag = sc.nextInt();
            if (flag == 0) {
                break;
            }
        }
        sc.close();
    }
}
```

**OUTPUT:**



```
        at QueueCheck.main(QueueCheck.java:63)
PS V:\DS> cd "v:\DS\" ; if ($?) { javac QueueCheck.java } ; if ($?) { java QueueCheck }
Enter the size of the queue: 4
Select an Operation:
 1) Enqueue
 2) Dequeue
 3) Front
 4) Rear
 5) Size
1
Enter the element to be enqueued: 1
Queue: [1]
Press 1 to continue or 0 to exit
1
Select an Operation:
 1) Enqueue
 2) Dequeue
 3) Front
 4) Rear
 5) Size
1
Enter the element to be enqueued: 2
Queue: [1,2]
Press 1 to continue or 0 to exit
1
Select an Operation:
 1) Enqueue
 2) Dequeue
 3) Front
 4) Rear
 5) Size
2
Dequeued element: 1
Queue: [2]
Press 1 to continue or 0 to exit
1
```



```
Dequeued element: 1
Queue: [2]
Press 1 to continue or 0 to exit
1
Select an Operation:
 1) Enqueue
 2) Dequeue
 3) Front
 4) Rear
 5) Size
3
Front element: 2
Press 1 to continue or 0 to exit
1
Select an Operation:
 1) Enqueue
 2) Dequeue
 3) Front
 4) Rear
 5) Size
4
Rear element: 2
Press 1 to continue or 0 to exit
1
Select an Operation:
 1) Enqueue
 2) Dequeue
 3) Front
 4) Rear
 5) Size
5
Size: 1
Queue: [2]
Press 1 to continue or 0 to exit
0
PS V:\DS>
```

| | |
|---|---|
| **CONCLUSION:** | In this experiment, I learned about various operations of queue such as Enqueue, Dequeue, Print, etc and created a menu-driven program to perform. |