Day 21: Implement Queue

Abhinav Yadav

"Queue: A way to handle the first task first."
— Anonymous

---

1 Introduction

A **Queue** is a linear data structure that follows the **First In First Out (FIFO)** principle. The first element inserted into the queue is the first one to be removed. It supports the following operations:

- **Enqueue:** Add an element to the rear of the queue.
- **Dequeue:** Remove the front element of the queue.
- **Peek/Front:** View the front element without removing it.

2 Applications of Queue

- Scheduling tasks in operating systems.
- Buffering data in streaming services.
- Managing requests in a web server.

3 Code

```
class Queue {
    private static final int MAX = 100; // Maximum size of the
       queue
    private int front, rear;
    private int[] data;

    // Constructor to initialize the queue
    public Queue() {
        front = -1;
        rear = -1;
        data = new int[MAX];
```

```
    }

    // Enqueue operation
    public void enqueue(int value) {
        if (rear == MAX - 1) {
            System.out.println("Queue Overflow");
            return;
        }
        if (front == -1) {
            front = 0;
        }
        data[++rear] = value;

        // Display the queue after the enqueue
        System.out.print("Queue after enqueue: ");
        for (int i = front; i <= rear; i++) {
            System.out.print(data[i] + " ");
        }
        System.out.println();
    }

    // Dequeue operation
    public int dequeue() {
        if (front == -1) {
            System.out.println("Queue Underflow");
            return -1;
        }
        int dequeuedValue = data[front];
        if (front == rear) {
            front = rear = -1;
        } else {
            front++;
        }

        // Display the queue after the dequeue
        System.out.print("Queue after dequeue: ");
        if (front != -1) {
            for (int i = front; i <= rear; i++) {
                System.out.print(data[i] + " ");
            }
        } else {
            System.out.print("Empty");
        }
        System.out.println();

        return dequeuedValue;
    }

    // Main function to test queue operations
    public static void main(String[] args) {
        Queue queue = new Queue(); // Initialize the queue
```

```
62
63          // Enqueue elements into the queue
64          queue.enqueue(10);
65          queue.enqueue(20);
66          queue.enqueue(30);
67
68          // Dequeue elements from the queue
69          System.out.println("Dequeued: " + queue.dequeue()); //
               Should print 10
70          System.out.println("Dequeued: " + queue.dequeue()); //
               Should print 20
71          System.out.println("Dequeued: " + queue.dequeue()); //
               Should print 30
72
73          // Try dequeuing from an empty queue
74          System.out.println("Dequeued: " + queue.dequeue()); //
               Should print "Queue Underflow"
75      }
76  }
```

4 Queue operations: Visual Representation and Output

Figure 1: Queue Operations: Enqueue and Dequeue

5 Conclusion

The queue data structure is crucial in applications where elements need to be processed in the order they arrive, such as in task scheduling and buffer management. It is a simple yet highly effective tool for implementing FIFO logic.

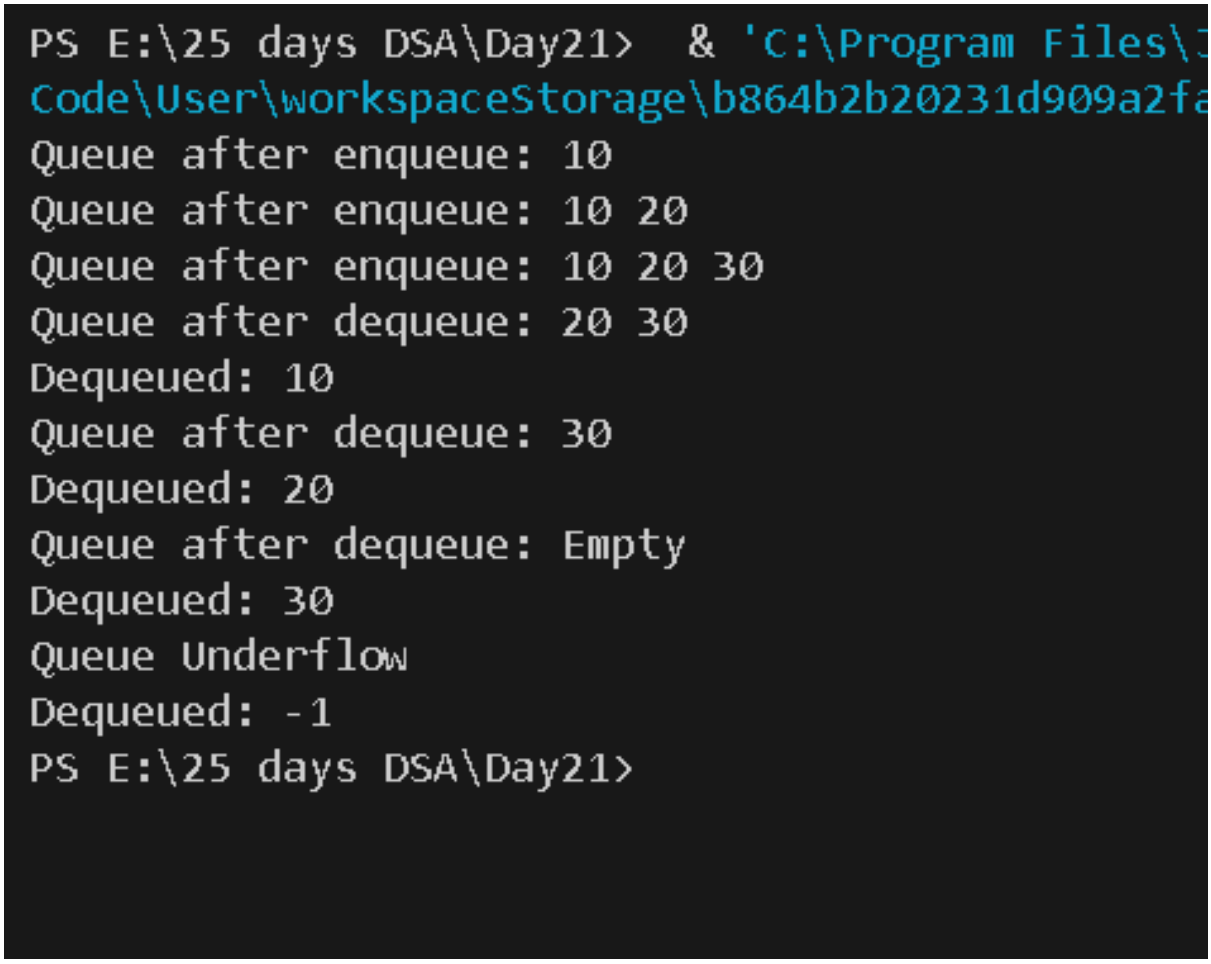

Figure 2: Program Output for Queue