Report on myArrayQueue Implementation

Introduction

The **MyArrayQueue** class is a circular queue built using an array-based structure. This approach optimizes memory usage while supporting dynamic resizing when the queue reaches its full capacity. The class implements two core operations: **enqueue**, which adds an element to the end of the queue, and **dequeue**, which removes the front element and returns it. Both operations have built-in exception handling to ensure robust functionality. Additionally, efficiency is a key aspect of this implementation, with most operations achieving constant-time complexity.

Algorithm & Implementation

Enqueue Operation:

Before inserting a new element, the queue checks whether it is full. If it has reached capacity, the underlying array is resized, typically doubling in size to accommodate additional elements. During this resizing process, all existing elements are repositioned to ensure proper ordering, with the front element moved to index 0. Once this restructuring is complete, the new element is added to the rear index, utilizing **circular indexing** to maintain efficient placement. This technique eliminates the need for costly element shifting, which would otherwise be necessary in a traditional linear array structure.

Dequeue Operation:

When performing a **dequeue** operation, the system first verifies whether the queue is empty. If there are no elements to remove, an **IllegalStateException** is thrown to signal an invalid operation. Once an element is removed from the front, the **front index** is updated according to circular indexing principles, ensuring smooth transitions between elements. To assist with garbage collection, the vacated position can be explicitly set to **null**, freeing memory and reducing unnecessary references.

Efficiency Analysis

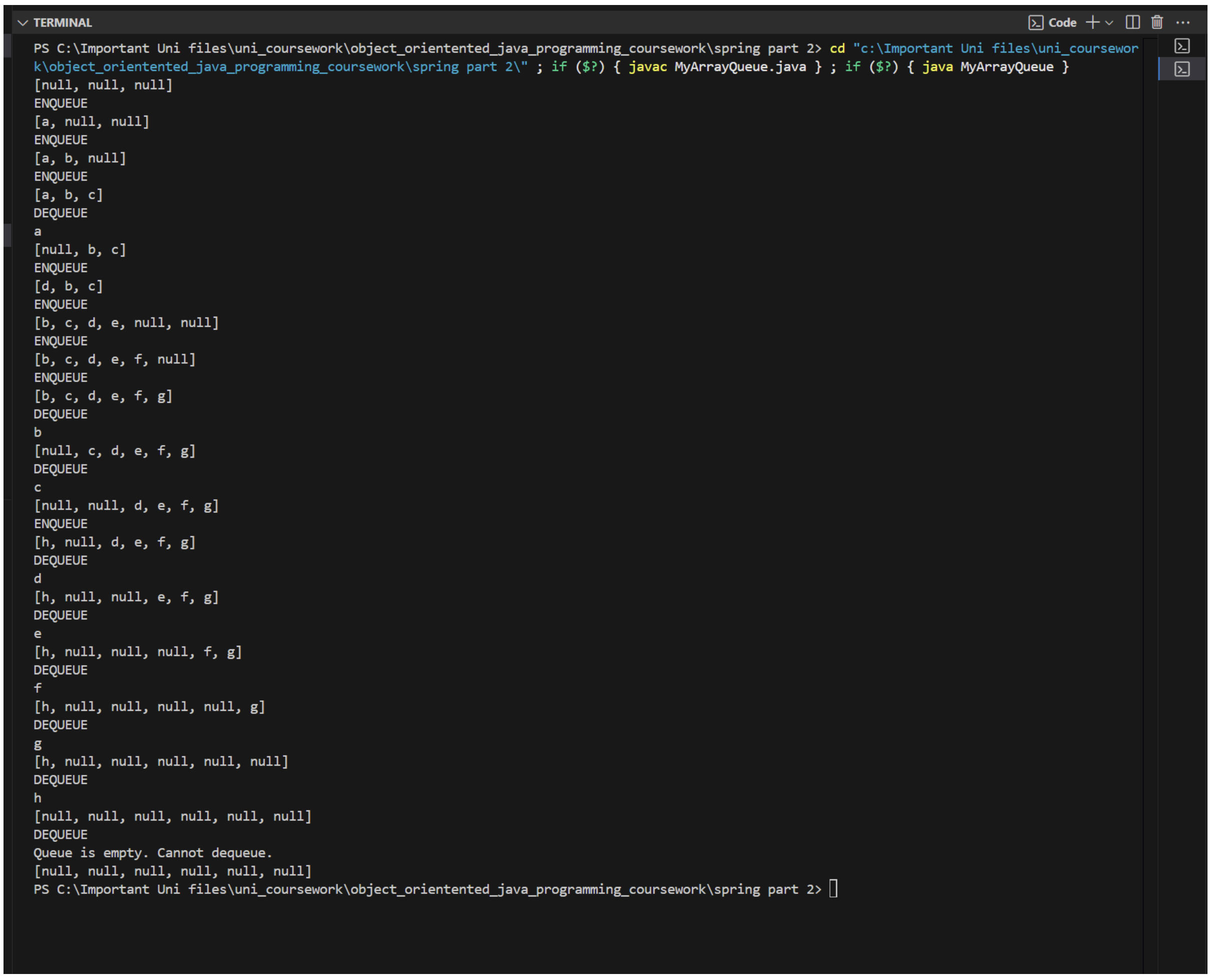
Time Complexity

The efficiency of the MyArrayQueue operations varies depending on the conditions. Normally, **enqueue** runs in constant time **O(1)** since it simply places the new element at the rear. However, if resizing is required, the operation takes **O(n)** time due to the need for copying all elements into a newly allocated array. The **dequeue** operation consistently runs in **O(1)** time, as removing the front element and updating the index is a direct operation.

Memory Considerations:

The **MyArrayQueue** class is designed to maximize memory efficiency while ensuring scalability. Dynamic resizing enables the queue to adapt to growing workloads without excessive memory overhead. Circular indexing further contributes to optimization by eliminating the need for frequent element shifting, making this approach more efficient compared to traditional array-based queue implementations.

Example Output Screenshot:



```
PS C:\Important Uni files\uni_coursework\object_orientented_java_programming_coursework\spring part 2> cd "c:\Important Uni files\uni_coursewor
k\object_orientented_java_programming_coursework\spring part 2\" ; if ($?) { javac MyArrayQueue.java } ; if ($?) { java MyArrayQueue }
[null, null, null]
ENQUEUE
[a, null, null]
ENQUEUE
[a, b, null]
ENQUEUE
[a, b, c]
DEQUEUE
a
[null, b, c]
ENQUEUE
[d, b, c]
ENQUEUE
[b, c, d, e, null, null]
ENQUEUE
[b, c, d, e, f, null]
ENQUEUE
[b, c, d, e, f, g]
DEQUEUE
b
[null, c, d, e, f, g]
DEQUEUE
c
[null, null, d, e, f, g]
ENQUEUE
[h, null, d, e, f, g]
DEQUEUE
d
[h, null, null, e, f, g]
DEQUEUE
e
[h, null, null, null, f, g]
DEQUEUE
f
[h, null, null, null, null, g]
DEQUEUE
g
[h, null, null, null, null, null]
DEQUEUE
h
[null, null, null, null, null, null]
DEQUEUE
Queue is empty. Cannot dequeue.
[null, null, null, null, null, null]
PS C:\Important Uni files\uni_coursework\object_orientented_java_programming_coursework\spring part 2> 
```

Conclusion

The **MyArrayQueue** class successfully balances efficiency and scalability through dynamic resizing and circular indexing. The constant-time complexity of enqueue and dequeue operations ensures fast performance, while built-in exception handling provides stability. Potential future enhancements could include an automatic shrinking mechanism to prevent excessive memory allocation when the queue experiences long periods of low utilization.