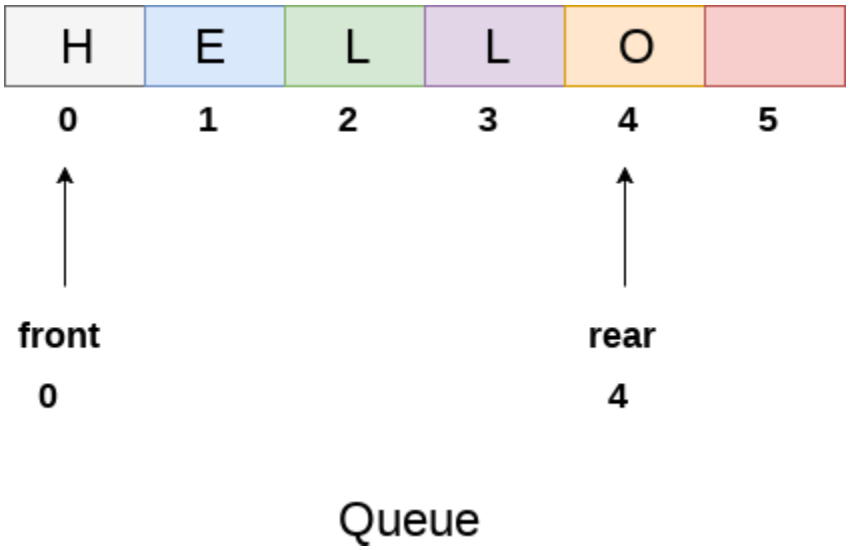
Implement Queue Using Array in Java

The queue is a type of data structure that can be implemented using an array or a linked list. Here, we have given a brief knowledge of the process of implementing a queue using an array.



Queue

Queue

A queue is data structure that is based on first-in first-out (FIFO) in which the first item input is also the first item removed. Items are added to the end of the line and removed from the beginning.

When utilising an array to construct a queue, the fact that an array has a fixed size once declared poses an issue in the queue implementation. When elements are added to a queue and then deleted, a gap is created. To fill the gap, we can rearrange the remaining components to fill the space, but it is a time-consuming procedure. Another alternative is to use a circular queue, with the front and back pointing to the beginning of the array after the maximum size has been achieved.

Queue Implementation in Java

Queue Operations

For a Queue-based system, the following three operations are used.

insert: To add an item at the back / rear end of the queue.

remove: To delete an item from the front of the queue.

peek: Get a value from the front of the queue without having to remove it.

Java Program for Queue Implementation Using an Array

DemoQueue.java

```
1.  public class DemoQueue
2.  {
3.    /* Member variable declaration */
4.    private int maxSize;
5.    private int[] queueArray;
6.    private int front;
7.    private int rear;
8.    private int currentSize;
9.    /* Constructor */
10.   public DemoQueue(int size)
11.   {
12.     this.maxSize = size;
13.     this.queueArray = new int[size];
14.     front = 0;
15.     rear = -1;
16.     currentSize = 0;
17.   }
18.   /* Queue:Insert Operation */
19.   public void insert(int item)
20.   {
21.     /* Checks whether the queue is full or not */
22.     if(isQueueFull())
```

```
23.    {
24.      System.out.println("Queue is full!");
25.      return;
26.    }
27.    if(rear == maxSize - 1)
28.{
29.      rear = -1;
30.    }
31.    /* increment rear then insert element to queue */
32.    queueArray[++rear] = item;
33.    currentSize++;
34.    System.out.println("Item added to queue: " + item);
35.  }
36.  /* Queue:Delete Operation */
37.  public int delete()
38.  {
39.    /* Checks whether the queue is empty or not */
40.    if(isQueueEmpty())
41.    {
42.      throw new RuntimeException("Queue is empty");
43.    }
44.    /* retrieve queue element then increment */
45.    int temp = queueArray[front++];
46.    if(front == maxSize)
47.    {
48.      front = 0;
49.    }
50.    currentSize--;
51.    return temp;
```

```
52.  }
53.  /* Queue:Peek Operation */
54.  public int peek()
55.  {
56.    return queueArray[front];
57.  }
58.  /* Queue:isFull Operation */
59.  public boolean isQueueFull()
60.  {
61.    return (maxSize == currentSize);
62.  }
63.  /* Queue:isEmpty Operation */
64.  public boolean isQueueEmpty()
65.{
66.    return (currentSize == 0);
67.  }
68.  /* Driver Code */
69.  public static void main(String[] args)
70.  {
71.    DemoQueue queue = new DemoQueue(10);
72.    queue.insert(2);
73.    queue.insert(3);
74.    System.out.println("Item deleted from queue: " + queue.delete());
75.    System.out.println("Item deleted from queue: " + queue.delete());
76.    queue.insert(5);
77.    System.out.println("Item deleted from queue: " + queue.delete());
78.  }
79.}
```

Output:

```
Item added to queue: 2
Item added to queue: 3
Item deleted from queue: 2
Item deleted from queue: 3
Item added to queue: 5
Item deleted from queue: 5
```

As you can see from the code there are insertions and deletions as the front and back progress towards the higher index. Though not implemented as a circular queue, it will result in the queue becoming full and unable to accept new items, even if space has been made in the front due to the deletions.

CurrentSize is a field that is increased with each entry and decremented with each removal to keep track of the current size. If maxSize = currentSize, the queue is genuinely full; otherwise, even if maxSize is reached, there remains room at the front that may be utilised to start from the beginning by wrapping the back. Similarly, after maxSize is reached, front can be wrapped to start from the beginning.

Performance of Queue

In a queue, items can be inserted and removed in O(1) time.