Code : LP_Code1.java

Output :

```
Elements of queue [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
removed element-0
[1, 2, 3, 4, 5, 6, 7, 8, 9]
head of queue-1
Size of queue-9
```

Q2. Write a program to implement the functionalities of a stack using the methods of queue only.
Solution :

Code : LP_Code2.java

Output :

```
current size: 6
The peek element is :
6
The peek element is :
5
The peek element is :
4
current size: 4
```

Approach :

- The idea behind this approach is to make one queue and push the first element in it.
- After the first element, we push the next element and then push the first element again and finally pop the first element.
- So, according to the FIFO rule of the queue, the second element that was inserted will be at the front and then the first element as it was pushed again later and its first copy popped out.
- So, this acts as a Stack and we do this at every step i.e. from the initial element to the second last element, and the last element will be the one that we are inserting and since we will be pushing the initial elements after pushing the last element, our last element becomes the first element.

Time Complexity:

- Push operation: O(N)
- Pop operation: O(1)

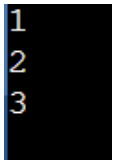
Space complexity: O(N) since 1 queue is used.

Q3. Write a program to implement the functionalities of a queue using the methods of stack only.

Solution :

Code : LP_Code3.java

Output :



Approach :

- Method 1 (By making enQueue/add operation costly): This method makes sure that the oldest entered element is always at the top of stack 1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used.

- enQueue(q, x):
- While stack1 is not empty, push everything from stack1 to stack2.
- Push x to stack1 (assuming size of stacks is unlimited).
- Push everything back to stack1.
- Here time complexity will be O(n)

- deQueue(q):
- If stack1 is empty then error
- Pop an item from stack1 and return it.

Time Complexity:

- Push operation: O(N).
- In the worst case we have an empty whole of stack 1 into stack 2.
- Pop operation: O(1).
- Same as pop operation in stack.

Space Complexity : O(N).
Use of stack for storing values.

Method 2 (By making deQueue operation costly):

- In this method, in en-queue operation, the new element is entered at the top of stack1. In de-queue operation, if stack2 is empty then all the elements are moved to stack2 and finally top of stack2 is returned.

- enQueue(q, x)
  1) Push x to stack1 (assuming size of stacks is unlimited).
  Here time complexity will be O(1)
- deQueue(q)
  1) If both stacks are empty then error.
  2) If stack2 is empty while stack1 is not empty, push everything from stack1 to stack2.
  3) Pop the element from stack2 and return it.
- Here time complexity will be O(n)

Code : LP_Code4.java

Output :