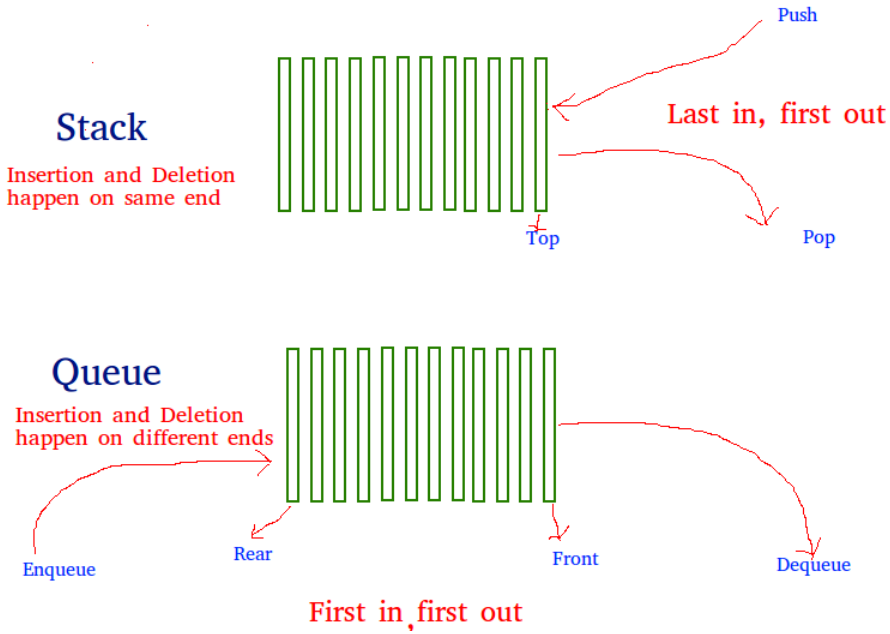
Implementing Queue using Stack

Problem: Given a stack data structure with push and pop operations, the task is to implement a queue using instances of stack data structure and operations on them.



Solution: *A queue can be implemented using two stacks.* Let the queue to be implemented be **q** and stacks used to implement **q** are **stack1** and **stack2** respectively.

The queue ***q*** can be implemented in two ways:

- **Method 1 (By making enQueue operation costly)**: This method makes sure that oldest entered element(element inserted first) is always at the top of stack1, so that deQueue operation just pops from stack1. To put the element at top of stack1, stack2 is used. The idea is to while pushing an element, first move all elements from stack1 to stack2, insert the new element to stack1 and then again move all elements from stack2 to stack1. Below is the implementation of both enQueue() and deQueue() operations:

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

deQueue(q)
  1) If stack1 is empty then print an error
  2) Pop an item from stack1 and return it
Here time complexity will be O(1)
```

Implementation:

C++ Java

```
// Java program to implement Queue using
// two stacks with costly enQueue()
import java.util.*;

class GFG
{
```

```
static class Queue
{
    static Stack<Integer> s1 = new Stack<Integer>();
    static Stack<Integer> s2 = new Stack<Integer>();

    static void enQueue(int x)
    {
        // Move all elements from s1 to s2
        while (!s1.isEmpty())
        {
            s2.push(s1.pop());
            //s1.pop();
        }

        // Push item into s1
        s1.push(x);

        // Push everything back to s1
        while (!s2.isEmpty())
        {
            s1.push(s2.pop());
            //s2.pop();
        }
    }

    // Dequeue an item from the queue
    static int deQueue()
    {
        // if first stack is empty
```

```
        if (s1.isEmpty())
        {
            System.out.println("Q is Empty");
            System.exit(0);
        }

        // Return top of s1
        int x = s1.peek();
        s1.pop();
        return x;
    }
};

// Driver code
public static void main(String[] args)
{
    Queue q = new Queue();
    q.enQueue(1);
    q.enQueue(2);
    q.enQueue(3);

    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
    System.out.println(q.deQueue());
}
}
```

Output:

```
1
2
3
```

- **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, the top of stack2 is returned. Below is the implementation of both enQueue() and deQueue() operations:

```
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)
```

Method 2 is better in performance than method 1. As Method 1 moves all the elements twice in enQueue operation, while method 2 (in deQueue operation) moves the elements once and moves elements only if stack2 is empty. **Implementation** :

C++ Java

```
// Java Program to implement a queue using two stacks

// Note that Stack class is used for Stack implementation
```

```java
import java.util.Stack;

public class GFG {
    /* class of queue having two stacks */
    static class Queue {
        Stack<Integer> stack1;
        Stack<Integer> stack2;
    }

    /* Function to push an item to stack*/
    static void push(Stack<Integer> top_ref, int new_data)
    {
        // Push the data onto the stack
        top_ref.push(new_data);
    }

    /* Function to pop an item from stack*/
    static int pop(Stack<Integer> top_ref)
    {
        /*If stack is empty then error */
        if (top_ref.isEmpty()) {
            System.out.println("Stack Underflow");
            System.exit(0);
        }

        // pop the data from the stack
        return top_ref.pop();
    }
```

```
// Function to enqueue an item to the queue
static void enQueue(Queue q, int x)
{
    push(q.stack1, x);
}

/* Function to deQueue an item from queue */
static int deQueue(Queue q)
{
    int x;

    /* If both stacks are empty then error */
    if (q.stack1.isEmpty() && q.stack2.isEmpty()) {
        System.out.println("Q is empty");
        System.exit(0);
    }

    /* Move elements from stack1 to stack 2 only if
        while (!q.stack1.isEmpty()) {
            x = pop(q.stack1);
            push(q.stack2, x);
        }
    }
    x = pop(q.stack2);
    return x;
}
```

```
    /* Driver function to test above functions */
    public static void main(String args[])
    {
        /* Create a queue with items 1 2 3*/
        Queue q = new Queue();
        q.stack1 = new Stack<>();
        q.stack2 = new Stack<>();
        enQueue(q, 1);
        enQueue(q, 2);
        enQueue(q, 3);

        /* Dequeue items */
        System.out.print(deQueue(q) + " ");
        System.out.print(deQueue(q) + " ");
        System.out.println(deQueue(q) + " ");
    }
}
// This code is contributed by Sumit Ghosh
```

Output:

```
1 2 3
```

Dash
All
Articles
Videos
Problems
Quiz

If you are facing any issue on this page. Please let us know.