STACK QUESTION

Question-> Queue using 2 stack

Code->

```
package Day53Queue;
import java.util.*;
public class Ques1QueueUsing2Stack {
    static class QueueUsing2Stack {
       static Stack<Integer> s1 = new Stack<>();
       static Stack<Integer> s2 = new Stack<>();

       public static boolean isEmpty(){
        return s1.isEmpty();
       }
       //add - O(n)
       public static void add(int data){
         // Move all elements from s1 to s2
        while(!s1.isEmpty()){
            s2.push(s1.pop());
        }
        // Push new element into s1
        s1.push(data);

        // Move everything back to s1 from s2
        while(!s2.isEmpty()){
            s1.push(s2.pop());
        }
       }
       //remove - O(1)
       public static int remove(){
        if(isEmpty()){
            System.out.println("Queue is empty");
            return -1;
        }
        return s1.pop();

       }
       //peek
       public static int peek(){
        if(isEmpty()){
            System.out.println("Queue is empty");
            return -1;
        }
        return s1.peek();
       }

    }
    public static void main(String[] args) {
        QueueUsing2Stack q = new QueueUsing2Stack();
```

STACK QUESTION

```
        q.add(1);
        q.add(2);
        q.add(3);
        while(!q.isEmpty()){
            System.out.println(q.peek());
            q.remove();
        }

    }


}
```

Pseudocode→

Class QueueUsing2Stack

Declare two stacks 's1' and 's2'

Method isEmpty()

Return true if 's1' is empty

Method add(data)

While 's1' is not empty

Move elements from 's1' to 's2' (s1.pop() → s2.push())

Push 'data' into 's1'

While 's2' is not empty

Move elements from 's2' back to 's1' (s2.pop() → s1.push())

Method remove()

If queue is empty (isEmpty() == true)

Print "Queue is empty"

Return -1

Else

Pop and return the top element from 's1'

```
Method peek()
  If queue is empty (isEmpty() == true)
    Print "Queue is empty"
    Return -1
  Else
    Return the top element from 's1'

Main method
  Create an instance of QueueUsing2Stack, 'q'
  Call q.add(1)
  Call q.add(2)
  Call q.add(3)

  While q is not empty (q.isEmpty() == false)
    Print q.peek()
    Call q.remove()
```

Explanation:

1. **Class Structure**: The `QueueUsing2Stack` class implements a queue using two stacks (`s1` and `s2`).
   - **s1**: Holds the elements in queue order.
   - **s2**: A temporary stack used to reverse the order of elements when adding a new element.
2. **isEmpty()**: This method checks if the queue is empty by checking if `s1` is empty. If `s1` is empty, the queue is empty.
3. **add(data)**:
   - This method adds a new element (`data`) to the queue.
   - It first moves all elements from `s1` to `s2` (to reverse the order).
   - Then, the new element is pushed onto `s1`.
   - Finally, all elements are moved back from `s2` to `s1`, restoring the queue order.
4. **remove()**:
   - This method removes and returns the front element of the queue by simply popping from `s1` (since the front element is at the top of `s1` after the add process).
   - If the queue is empty, it returns `-1`.
5. **peek()**:
   - This method returns the front element of the queue without removing it by checking the top of `s1`. If `s1` is empty, it returns `-1`.
6. **Main Method**:

- A `QueueUsing2Stack` object (`q`) is created.
- The elements `1`, `2`, and `3` are added to the queue.
- Then, a loop runs while the queue is not empty, printing the front element (`q.peek()`) and removing it (`q.remove()`).

This code demonstrates how to implement a queue with two stacks, where the enqueue operation (`add`) has a time complexity of O(n)O(n)O(n) and the dequeue operation (`remove`) has a time complexity of O(1)O(1)O(1).

4o

Algorithm for Queue Using Two Stacks:

1. isEmpty()

- **Input**: None
- **Output**: Returns `true` if the queue is empty, `false` otherwise
- **Steps**:
  1. Check if `s1` is empty.
  2. If `s1` is empty, return `true`.
  3. Else, return `false`.

2. add(data)

- **Input**: `data` (the element to be added to the queue)
- **Output**: None
- **Steps**:
  1. While `s1` is not empty:
     - Pop the top element from `s1` and push it onto `s2`.
  2. Push the new element (`data`) onto `s1`.
  3. While `s2` is not empty:
     - Pop the top element from `s2` and push it back onto `s1`.

3. remove()

- **Input**: None
- **Output**: Returns the front element of the queue or `-1` if the queue is empty
- **Steps**:
  1. If the queue is empty (call `isEmpty()`):
     - Print "Queue is empty" and return `-1`.
  2. Otherwise, pop the top element from `s1` and return it.

4. peek()

- **Input**: None
- **Output**: Returns the front element of the queue or `-1` if the queue is empty
- **Steps**:
  1. If the queue is empty (call `isEmpty()`):
     - Print "Queue is empty" and return `-1`.
  2. Otherwise, return the top element from `s1`.

5. Main Method

- **Input**: None
- **Output**: Prints the elements at the front of the queue while removing them
- **Steps**:
    1. Create a `QueueUsing2Stack` object `q`.
    2. Call `q.add(1)`, `q.add(2)`, and `q.add(3)` to add elements to the queue.
    3. While `q.isEmpty()` returns `false`:
        - Call `q.peek()` to print the front element.
        - Call `q.remove()` to remove the front element.

This algorithm follows a **two-stack approach** for simulating the queue where the `add` operation involves shifting elements between the two stacks to maintain order, and `remove` is efficient with a direct pop from the stack.

Question-> Stack Using 2 Queue?

Code-