Task 04:

Stack Sorting In-Place

You must write a function to sort a stack such that the smallest items are on the top. You can use an additional temporary stack, but you may not copy the elements into any other data structure such as an array. The stack supports the following operations: push, pop, peek, and isEmpty.

Function: -

```
package com.wipro.assignment;

public class Stack {


    private int[] items;

    private int top;


    public Stack() {

        items = new int[10]; // Initial capacity, can be
adjusted

        top = -1;

    }
```

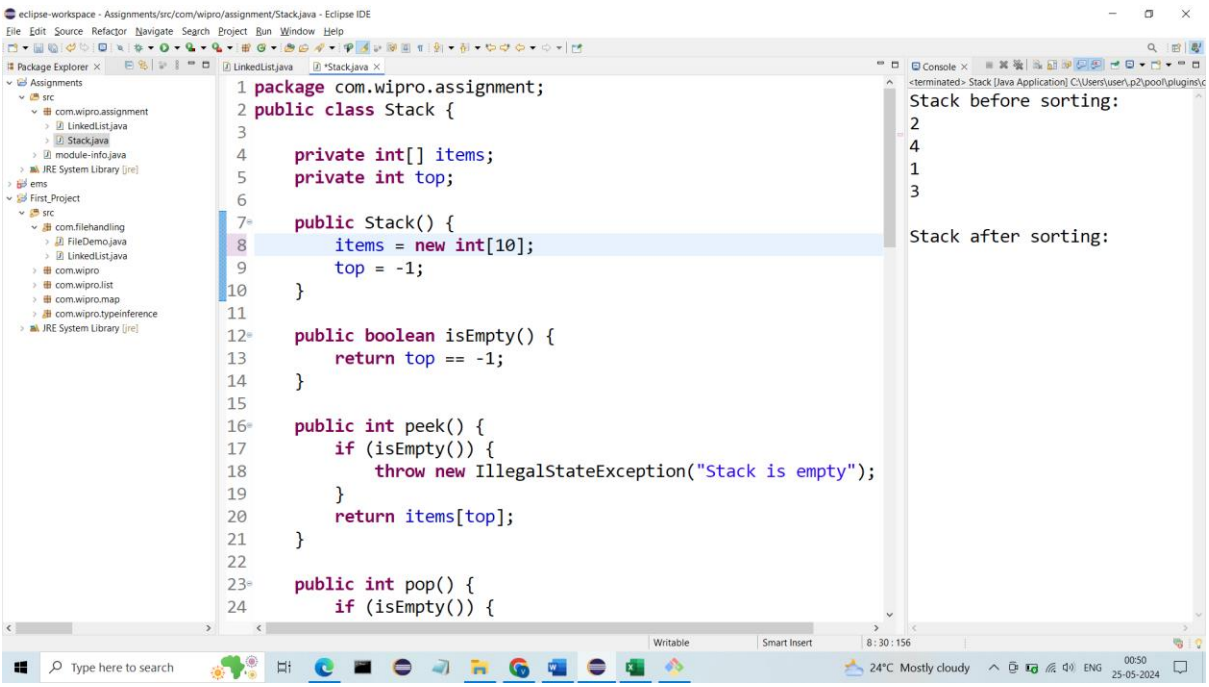
```
    public boolean isEmpty() {
        return top == -1;
    }

    public int peek() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is
empty");
        }
        return items[top];
    }

    public int pop() {
        if (isEmpty()) {
            throw new IllegalStateException("Stack is
empty");
        }
        return items[top--];
    }

    public void push(int item) {
```

```
        if (top == items.length - 1) {
            // Resize if needed
            int[] newItems = new int[items.length * 2];
            System.arraycopy(items, 0, newItems, 0,
items.length);
            items = newItems;
        }
        items[++top] = item;
    }

    public static void sortStack(Stack s) {
        Stack temp = new Stack();

        while (!s.isEmpty()) {
            int tempVal = s.pop();

            while (!temp.isEmpty() && temp.peek() <
tempVal) {
                s.push(temp.pop());
            }
```

```
            temp.push(tempVal);
        }

        // Re-stack elements from temp to original stack
(now sorted)
        while (!temp.isEmpty()) {
            s.push(temp.pop());
        }
    }

    public static void main(String[] args) {
        Stack myStack = new Stack();
        myStack.push(3);
        myStack.push(1);
        myStack.push(4);
        myStack.push(2);

        System.out.println("Stack before sorting:");
        while (!myStack.isEmpty()) {
            System.out.println(myStack.pop());
```

```
        }

        sortStack(myStack);

        System.out.println("\nStack after sorting:");
        while (!myStack.isEmpty()) {
            System.out.println(myStack.pop());
        }
    }
}
```

Output: -



Explanation:

1. **Stack Class:** This implements the basic Stack functionalities like push, pop, peek, and isEmpty using an underlying array. It also includes logic for resizing the array if needed.
2. **sortStack Function:**
    - It takes the stack s to be sorted as input.
    - It creates an empty temporary stack temp.
    - It iterates through the elements of s using a while loop until s is empty:
        - It pops an element (tempVal) from s.
        - It uses another while loop to insert tempVal in the correct position in the temporary stack (temp):
            - It keeps popping elements from temp and pushing them back to s as long as the element at the top of temp is less than tempVal.
            - This ensures that elements smaller than tempVal are placed below it in temp.
        - Finally, it pushes tempVal onto temp.

- After processing all elements from s, temp will contain the elements in sorted order (smallest on top).
- To move the sorted elements back to s, the function iterates through temp and pops elements back to s (which effectively reverses the order in temp).

Time Complexity: O(n^2) in the worst case, where n is the number of elements in the stack.

Space Complexity: O(n), as we use an additional temporary stack for sorting.