

1. Consider the following class:

```
public class IntStack
{
    private int[] items;
    private int sp;

    public IntStack()
    {
        items = new int[1024];
        sp = 0;
    }

    public int peek()
    {
        if (sp == 0)
            throw new NoSuchElementException();
        return items[sp - 1];
    }

    < ... other methods not shown >
}
```

Write the `isEmpty`, `push`, and `pop` methods.

2. Write a method

```
public void rearrange(Stack<Integer> stk)
```

that rearranges the values on `stk` in such a way that all positive values are on top, in the same order, followed by all zeroes, followed by all negative values in the same order, on the bottom.

3. Suppose a queue is implemented as a circular doubly-linked list with a header node:

```
public class MyQueue
{
    private ListNode2 header;

    public MyQueue()
    {
        header = new ListNode2(null, null, null);
        header.setNext(header);
        header.setPrevious(header);
    }

    public Object peek()
    {
        if (header.getNext() == header)
            throw new NoSuchElementException();
        return header.getNext().getValue();
    }

    < ... other methods not shown >
}
```

Write an `add` method.

4. ■ A class `IntQueue` represents a queue of ints. Suppose its `add` method is coded as follows:

```
public boolean add(int x)
{
    int temp = rear + 1;
    if (temp >= items.length)
        temp = 0;
    if (temp == front)
        return false;
    items[rear] = x;
    rear = temp;
    return true;
}
```

Write a `remove` method:

```
public int remove()
```

5. Write a method

```
// Merges the strings from q1 and q2 into q, in alphabetical
// order.
// Precondition: q1 and q2 are sorted in alphabetical order;
//               q is empty.
public static void merge(Queue<String> q1, Queue<String> q2,
                        Queue<String> q)
```

6. ■ Postfix notation is a way of writing algebraic expressions with no parentheses. For example, instead of writing  $a + b$ , in postfix form we would write  $a b +$ . Instead of  $(a + b) * (c + d)$ , we would write  $a b + c d + *$ .

- (a) Write and test a method

```
public String eval(Queue<String> q)
```

that evaluates a postfix expression for concatenating strings. An expression is represented as a queue of strings in which some strings are operands and other strings are equal to "+", which signifies the concatenation operation. Use the following algorithm:

1. Create an empty stack.
2. If the next element in the queue is an operand (not a "+"), push it onto the stack; if it is a "+", pop the last two operands in the appropriate order from the stack and push their "sum" back on the stack.

If the expression is formed correctly, the stack is never empty when you need to pop a value and there is one string left on the stack at the end. This is the result of `eval` that it should be returned. If the expression is not formed correctly, your method should return `null`.

- (b) If + is the only type of operation in the expression, a simpler method will work, too. Write an alternative `eval` method, which simply concatenates all the strings in `q` in order, skipping all the "+" strings.