Name: zhenzhen su

Class: CSC 143

Instructor: Dr Reid

1. Problem statement: write a method splitStack to put negative number to bottom of stack and non-negative number to top of stack

Code:

```
public static void main(String[] args) {
    //call splitStack to put negatives number to bottom of stack
    Stack<Integer> s = new Stack<Integer>();
    s.push(3);
    s.push(-5);
    s.push(1);
    s.push(2);
    s.push(-4);
    System.out.println("The stack is " + s);
    System.out.println("The stack is after calling splitstack method: " +
splitStack(s));
}

//splitStack put negatives to the bottom of stack and non-negatives to the top of
stack
public static Stack<Integer> splitStack (Stack<Integer> s){
    int size = s.size();
    Queue<Integer> q = new LinkedList<Integer>();
    //pop elements in stack and put into queue
    for(int i = 0; i < size; i++){
        q.add(s.pop());
    }

    //put negative elements from queue into stack
    for(int i = 0; i < size; i++){
        int n = q.remove();
        if(n < 0){
            s.push(n);
        }else{
            q.add(n);
        }
    }

    //put the rest elements in queue into stack
    while(!q.isEmpty()){
        s.push(q.remove());
    }
    return s;
}
```

Console Output:

```
The stack is [3, -5, 1, 2, -4]
The stack is after calling splitstack method: [-4, -5, 2, 1, 3]
```

2. Problem: write a method called switchPairs to switch neighboring pairs of numbers starting from bottom of stack

Code:

```
public static void main(String[] args) {
        Stack<Integer> s = new Stack<Integer>();
        s.push(3);
        s.push(-5);
        s.push(1);
        s.push(2);
        s.push(-4);

        Stack<Integer> s1 = new Stack<Integer>();
        s1.push(3);
        s1.push(-5);
        s1.push(1);
        s1.push(2);
        System.out.println("Odd number of elements in the stack" + s);
        System.out.println("After method call gives : " + switchPairs(s));
        System.out.println("Even number of elements in the stack" + s1);
        System.out.println("After method call gives : " + switchPairs(s1));

}

    //switch two elements in a neighboring pair starting at the bottom of stack
    //postcon: stack contains the same elements as input stack
    public static Stack<Integer> switchPairs (Stack<Integer> s){
        int size = s.size();
        int index = 0;
        Queue<Integer> q = new LinkedList<Integer>();
        //for odd number of elements, put the top element to queue
        if(s.size() % 2 == 1){
            q.add(s.pop());
        }
        //pop elements in stack and put into queue
        //switch pairs when inserting into queue
        while(!s.isEmpty()){
            int n = s.pop();
            q.add(s.pop());
            q.add(n);
        }
        //put elements of queue back to stack but elements are in reverse order
        s = queueToStack(q);
        //put stack elements to queue and reverse that operation
        //to put elements in the original order
        q = StackToQueue(s);
        s = queueToStack(q);
        return s;
    }

    //transfer elements in queue into stack one by one
    public static Stack<Integer> queueToStack(Queue<Integer> q){
        Stack<Integer> s = new Stack<Integer>();
        while (!q.isEmpty()) {
            s.push(q.remove());
        }
        return s;
    }

    //transfer elements in stack to queue
```

```
    public static Queue<Integer> StackToQueue(Stack<Integer> s){
        Queue<Integer> q = new LinkedList<>();
        while (!s.isEmpty()) {
            q.add(s.pop());
        }
        return q;
    }
```

Console Output:

```
Odd number of elements in the stack[3, -5, 1, 2, -4]
After method call gives : [-5, 3, 2, 1, -4]
Even number of elements in the stack[3, -5, 1, 2]
After method call gives : [-5, 3, 2, 1]
```

3. Problem statement: reverse the first k elements of queue, leave the rest in order. If k is zero or negative, queue doesn't change. If k is bigger than the size of the queue, throw IllegalArgumentException.

```
public static void main(String[] args) {
        //reverseFirstK
        Queue<Integer> q = new LinkedList<Integer>();
        q.add(10);
        q.add(20);
        q.add(30);
        q.add(40);
        q.add(50);
        q.add(60);
        q.add(70);
        q.add(80);
        System.out.println("reverse first five elements " + reverseFirstK(5,q));
        System.out.println("reverse first zero elements " + reverseFirstK(0,q));
        System.out.println("reverse first -5 elements " + reverseFirstK(-5,q));
        System.out.println("reverse first eleven elements " + reverseFirstK(11,q));
}

//reverseFirstK
public static Queue<Integer> reverseFirstK (int k, Queue<Integer> q){
    Stack<Integer> s = new Stack<Integer>() ;
    int size = q.size();
    if(k > q.size()){
        throw new IllegalArgumentException();
    }else if(k > 0){
        //put first k element into a stack
        for(int i = 0; i < k; i++){
            s.push(q.remove());
        }
        //put elements in stack back to queue
        while(!s.isEmpty()){
            q.add(s.pop());

        }

        for(int i = 0; i < size - k; i++){
            q.add((q.remove()));
        }
```

```
    }
    return q;
}
```

Console output:

```
The queu is [10, 20, 30, 40, 50, 60, 70, 80]
reverse first five elements [50, 40, 30, 20, 10, 60, 70, 80]
reverse first zero elements [50, 40, 30, 20, 10, 60, 70, 80]
reverse first -5 elements [50, 40, 30, 20, 10, 60, 70, 80]
Exception in thread "main" java.lang.IllegalArgumentException
```

4. Problem statement: write a method to check if the stack is in ascending order from top to bottom and return Boolean variable

Code:

```
public static void main(String[] args) {
        //isSorted
        Stack<Integer> stack = new Stack<Integer>();
        stack.push(20);
        stack.push(20);
        stack.push(17);
        stack.push(11);
        stack.push(8);
        stack.push(8);
        stack.push(3);
        stack.push(2);

        Stack<Integer> stack1 = new Stack<Integer>();
        stack1.push(1);

        Stack<Integer> stack2 = new Stack<Integer>();
        stack2.push(20);
        stack2.push(-5);
        stack2.push(17);
        stack2.push(11);

        Stack<Integer> stack3 = new Stack<Integer>();

        System.out.println("stack is sorted ? " + isSorted(stack));
        System.out.println("stack1 is sorted ?" + isSorted(stack1));
        System.out.println("stack2 is sorted ? " + isSorted(stack2));
        System.out.println("stack3 is sorted ?" + isSorted(stack3));
}



//check if the stack is sorted from largest to smallest from bottom to top
public static boolean isSorted (Stack<Integer> s){
    boolean flag = true;
    Stack<Integer> s1 = new Stack<Integer>();
    //check if there is element
    //if yes, push the top element to auxiliary stack
    if(!s.isEmpty()){
        s1.push(s.pop());
    }
    //loop through stack, and compare elements
    while(!s.isEmpty() && flag) {
        int m = s1.pop();
        int n = s.pop();
```

```
            //compare current element at the top of stack to the element
            // before it
            if (n >= m) {
                //push elements in the stack to the auxiliary stack
                s1.push(m);
                s1.push(n);
            } else {
                flag = false;
                s1.push(m);
                s1.push(n);
            }
        }

        //restore the elements back to stack from the auxiliary stack
        while(!s1.isEmpty()){
            s.push(s1.pop());
        }
        System.out.println(s);
        return flag;
    }
```

Output console:

[20, 20, 17, 11, 8, 8, 3, 2]

stack is sorted ? true

[1]

stack1 is sorted ?true

[20, -5, 17, 11]

stack2 is sorted ? false

[]

stack3 is sorted ?true

Problem 5: write a method called interleave alternates elements from first half of queue with second half of the queue. The method throws an IllegalArugumentException.

Code:

```
public static void main(String[] args) {

        //interleave
        Queue<Integer> q = new LinkedList<Integer>();
        q.add(10);
        q.add(20);
        q.add(30);
        q.add(40);
        q.add(50);
        q.add(60);
        q.add(70);
```

```
        q.add(80);
        q.add(100);

        Queue<Integer> q1 = new LinkedList<Integer>();
        q1.add(10);
        q1.add(20);
        q1.add(30);
        q1.add(40);
        q1.add(50);
        q1.add(60);
        System.out.println("The queue is " + q1);
        System.out.println("The interleave is " + interleave(q1));
        System.out.println("The queue is " + q);
        System.out.println(interleave(q));
}

//interleave alternates elements from first half of queue with second half of the
queue
public static Queue<Integer> interleave(Queue<Integer> q){
    int size = q.size();
    Stack<Integer> s = new Stack<Integer>();

    //check the queue has a even size
    if(q.size() % 2 == 0){
        //put elements in stack
        while(!q.isEmpty()){
            s.push(q.remove());
        }

        //put top half elements of stack back to queue
        for(int i = 0; i < size/2 ; i++){
            q.add(s.pop());
        }

        //combine stack and queue elements with interleave
        for(int i = 0; i < size/2 ; i++){
            q.add(q.remove());
            q.add(s.pop());
        }

        //put elements to stack
        s = queueToStack(q);
        //put element back to queue;
        q = StackToQueue(s);

    }else{
        throw new IllegalArgumentException();
    }
    return q;
}
```

Console output:

The queue is [10, 20, 30, 40, 50, 60]
The interleave is [10, 40, 20, 50, 30, 60]
Exception in thread "main" java.lang.IllegalArgumentException
The queue is [10, 20, 30, 40, 50, 60, 70, 80, 100]