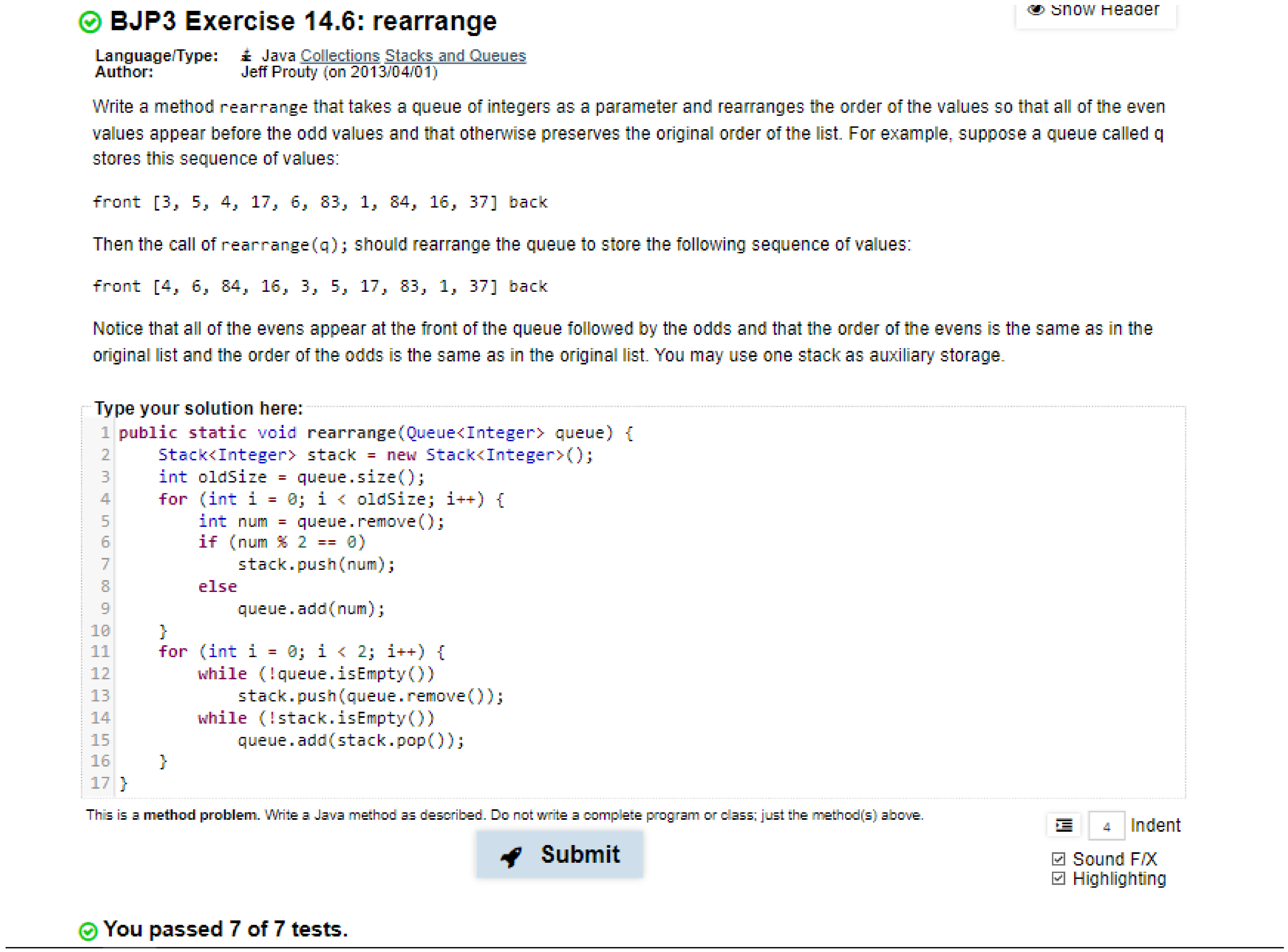


BJP3 Exercise 14.6: rearrange

Language/Type: Java Collections Stacks and Queues
Author: Jeff Prouty (on 2013/04/01)

Write a method `rearrange` that takes a queue of integers as a parameter and rearranges the order of the values so that all of the even values appear before the odd values and that otherwise preserves the original order of the list. For example, suppose a queue called q stores this sequence of values:

```
front [3, 5, 4, 17, 6, 83, 1, 84, 16, 37] back
```

Then the call of `rearrange(q);` should rearrange the queue to store the following sequence of values:

```
front [4, 6, 84, 16, 3, 5, 17, 83, 1, 37] back
```

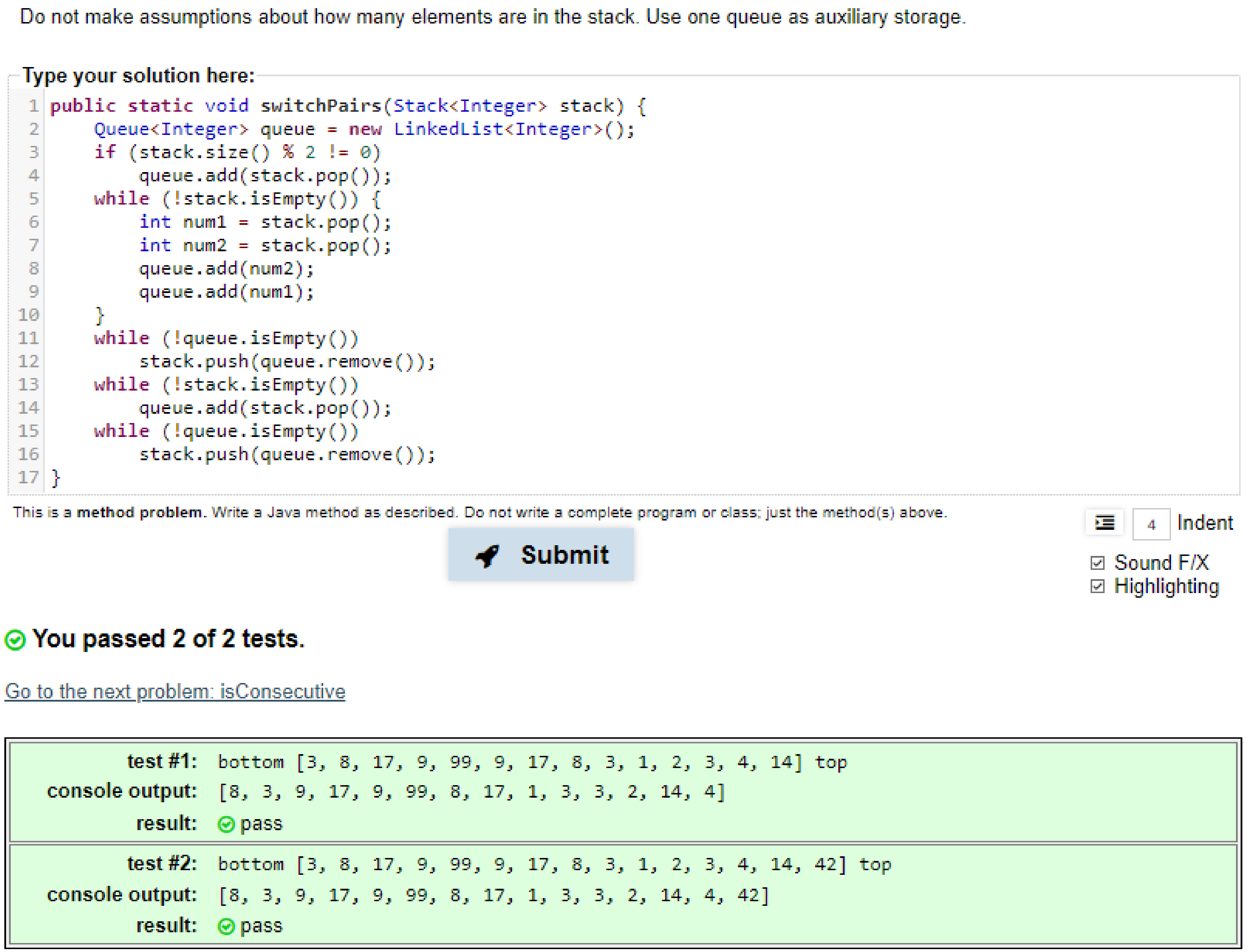
Notice that all of the evens appear at the front of the queue followed by the odds and that the order of the evens is the same as in the original list and the order of the odds is the same as in the original list. You may use one stack as auxiliary storage.

Type your solution here:

```java
public static void rearrange(Queue<Integer> queue) {
    Stack<Integer> stack = new Stack<Integer>();
    int oldSize = queue.size();
    for (int i = 0; i < oldSize; i++) {
        int num = queue.remove();
        if (num % 2 == 0)
            stack.push(num);
        else
            queue.add(num);
    }
    for (int i = 0; i < 2; i++) {
        while (!queue.isEmpty())
            stack.push(queue.remove());
        while (!stack.isEmpty())
            queue.add(stack.pop());
    }
}
```

This is a **method problem**. Write a Java method as described. Do not write a complete program or class; just the method(s) above.

You passed 7 of 7 tests.



Do not make assumptions about how many elements are in the stack. Use one queue as auxiliary storage.

Type your solution here:

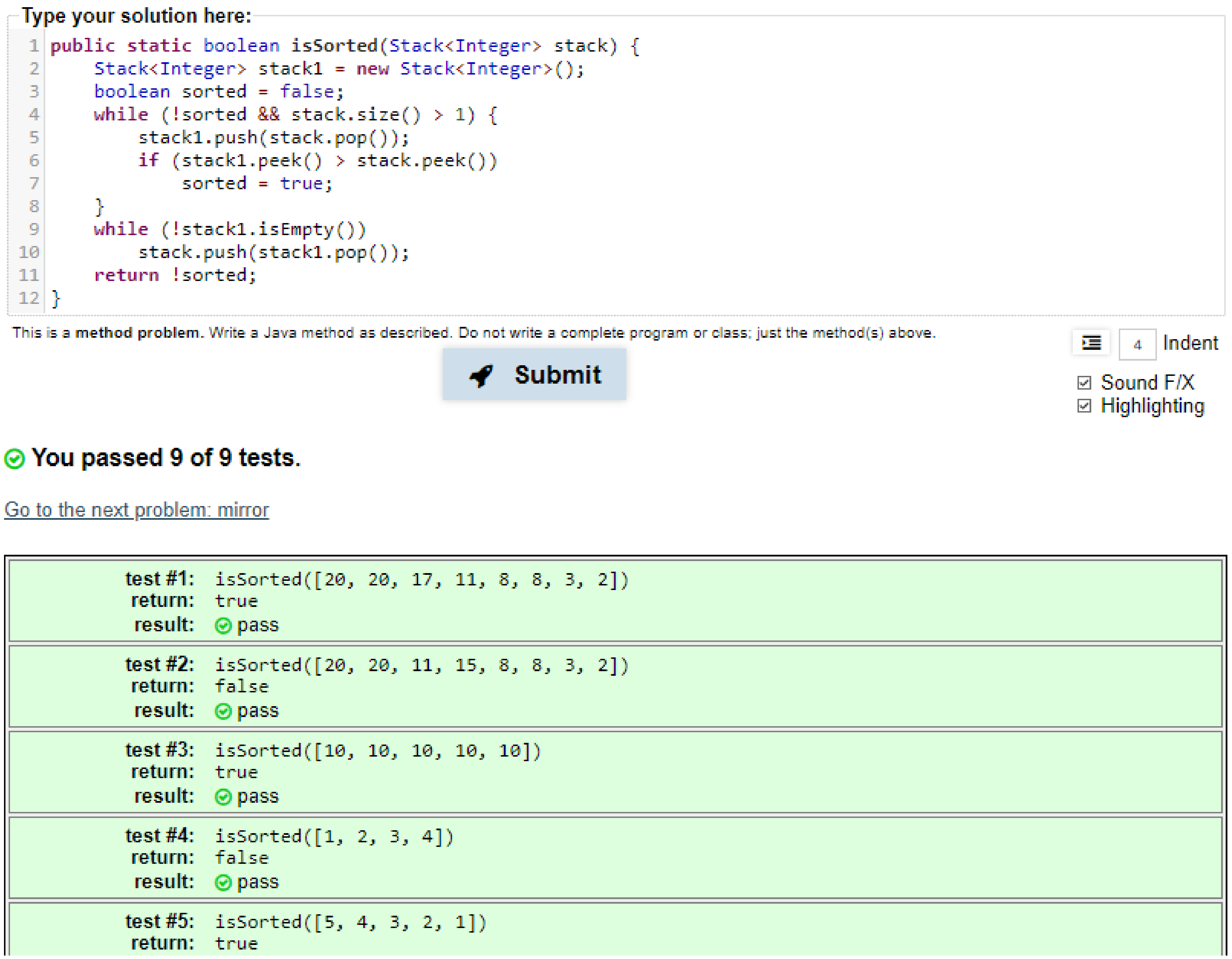
```java
public static void switchPairs(Stack<Integer> stack) {
    Queue<Integer> queue = new LinkedList<Integer>();
    if (stack.size() % 2 != 0)
        queue.add(stack.pop());
    while (!stack.isEmpty()) {
        int num1 = stack.pop();
        int num2 = stack.pop();
        queue.add(num2);
        queue.add(num1);
    }
    while (!queue.isEmpty())
        stack.push(queue.remove());
    while (!stack.isEmpty())
        queue.add(stack.pop());
    while (!queue.isEmpty())
        stack.push(queue.remove());
}
```

This is a **method problem**. Write a Java method as described. Do not write a complete program or class; just the method(s) above.

You passed 2 of 2 tests.

Go to the next problem: isConsecutive

test #1:	`bottom [3, 8, 17, 9, 99, 9, 17, 8, 3, 1, 2, 3, 4, 14] top`
console output:	`[8, 3, 9, 17, 9, 99, 8, 17, 1, 3, 3, 2, 14, 4]`
result:	✓ pass
test #2:	`bottom [3, 8, 17, 9, 99, 9, 17, 8, 3, 1, 2, 3, 4, 14, 42] top`
console output:	`[8, 3, 9, 17, 9, 99, 8, 17, 1, 3, 3, 2, 14, 4, 42]`
result:	✓ pass



Type your solution here:

```java
public static boolean isSorted(Stack<Integer> stack) {
    Stack<Integer> stack1 = new Stack<Integer>();
    boolean sorted = false;
    while (!sorted && stack.size() > 1) {
        stack1.push(stack.pop());
        if (stack1.peek() > stack.peek())
            sorted = true;
    }
    while (!stack1.isEmpty())
        stack.push(stack1.pop());
    return !sorted;
}
```

This is a **method problem.** Write a Java method as described. Do not write a complete program or class; just the method(s) above.

4 Indent

☑ Sound F/X
☑ Highlighting

Submit

You passed 9 of 9 tests.

Go to the next problem: mirror

test #1:	`isSorted([20, 20, 17, 11, 8, 8, 3, 2])`
return:	`true`
result:	✓ pass
test #2:	`isSorted([20, 20, 11, 15, 8, 8, 3, 2])`
return:	`false`
result:	✓ pass
test #3:	`isSorted([10, 10, 10, 10, 10])`
return:	`true`
result:	✓ pass
test #4:	`isSorted([1, 2, 3, 4])`
return:	`false`
result:	✓ pass
test #5:	`isSorted([5, 4, 3, 2, 1])`
return:	`true`