

Half-Exam 1 Feedback

Name:

Points ①-③ are things I would not take off points for on a quiz, but I recommend you learn to apply them both in the lab and on the quiz because thinking like this helps you write code faster and with fewer bugs.

Grading rubric:

(n) indicates number of points for this check-point.

(3) Check that exception thrown if $\text{index} < 0 \ || \ \text{index} \geq \text{size}$ [Or $\text{size} - 1 > \text{index}$]. (Note that $\text{index} < 0$ MAY be handled correctly automatically --- but better to put it in the condition)

(3) Throw exception on the condition being checked. See ⑩-⑪.

(5) Do not allocate new array, or allocate new array of correct size with correct syntax. See ④ and ⑤. -3 for omitting length.

(5) Return the previous element without overwriting it first. See ⑦.

(4) Ensure all variables (including the result) are initialized.

(10) Construct a correct loop or loops that walks through the elements that need to be moved. **I usually make this worth more points (roughly 30-60% of the total points for a problem). -3 if off by one.**

(5) Copy the elements with the correct indices after the index and before the index if accidentally done.

(5) Set the array at $\text{array}[\text{size}-1]$ to null. See ⑧. -3 if doesn't work with loop above.

(5) Decrement the size variable.

(5) Use correct array syntax.

Problem 1

① Reduce the complexity of loops: When an if block inside a loop is only executed once, you can eliminate the if statement and pull the block outside of the loop. Instead of writing:

```
public E remove2(int index) {
    /* check for out of bounds ... */
    E result = null;
    for(int oldInd = index; oldInd < size; oldInd++) {
        if(oldInd == index) {
            // This only executes once. Let's move it out.
            result = array[oldInd];
        } else if(oldInd == size-1) {
            // This only executes once. Let's move it out.
            array[oldInd] = null;
        } else {
            array[oldInd-1] = array[oldInd];
        }
    }
    size--;
    return result;
}
```

Note how much simpler the code becomes:

```
@Override
public E remove(int index) {
    /* check for out of bounds ... */
    E result = array[index];
    for(int oldInd = index; oldInd < size-1; oldInd++) {
        array[oldInd-1] = array[oldInd];
    }
    array[size-1] = null;
    size--;
    return result;
}
```

This principle simplifies the code even more when you DO need to move the data before and after the middle – see ③

② There is no need to move the data before the index.

③ If you choose to move the data before the index, principle ① is even more important. Instead of

```
public E remove(int index) {
    /* check for out of bounds ... */
    int newInd = 0;
    E result = null;
    for(int oldInd = 0; oldInd < size; oldInd++) {
        if(oldInd == index) {
            // This only executes once. Let's move it out.
            result = array[oldInd];
        } else if(oldInd == size-1) {
            // This only executes once. Let's move it out.
            array[oldInd] = null;
        } else { // before and after index
            // Two variables are needed here because the code
            // does different things before and after the index.
            // Let's eliminate one of them.
            array[newInd] = array[oldInd];
            newInd++;
        }
    }
    size--;
    return result;
}
```

You can write:

```
public E remove4(int index) {
    /* check for out of bounds ... */
    for(int oldInd = 0; oldInd < index; oldInd++) {
        // Hey! this does nothing! Let's eliminate it!
        array[oldInd] = array[oldInd];
    }
    E result = array[index];
    for(int oldInd = index+1; oldInd < size; oldInd++) {
        array[oldInd-1] = array[oldInd];
    }
    array[size-1] = null;
    size--;
    return result;
}
```

Notice we can eliminate a local variable in the process – simplifying our thinking about the code. Then we notice that the first loop doesn't do anything (in this case), so we can eliminate it to – one less loop to debug. (Sometimes that first loop IS useful – e.g., when we must resize an array. Even then, two loops are often simpler to check than one loop with an extra variable.)

④ There is no need to allocate a new array. In fact, allocating the array can make the operation significantly slower – going to an $O(n)$ operation from what can be an $O(1)$. Whenever we allocate a new array, we need to copy all the data to the new array. This is an $O(n)$ operation (where n is the

size of the list), because we must go through all the data. On the other hand, if I call `remove(n-1)`, there is no need to shift the elements after the removed item (because there aren't any!) so there is no need to go through a loop at all, and it is only an $O(1)$ operation (See the second code example in ① for an example of a loop that you wouldn't need to go through.)

⑤ If you allocate an array it SHOULD have the same size as the original array.

⑥

⑦ We need to save the value in the current spot in a variable so we can return it. See ①-③ for examples of this.

⑧ There is no need to iterate through the whole array. As long as we keep the unused elements null, we only need to set the value at `array[size - 1]` to null and then decrement the size. It is true that you COULD copy all the nulls over through `array.length`, but this is wasteful, since they are already null. Remember that the null values in indices less than size are elements in the list. They have their own index.

⑨

⑩ -2 To handle the index out of bounds case, we throw an exception to the method that calls us. There is no need to catch the exception because we can detect with a simple if statement that there is a problem. Rather than writing

```
try {  
  
} catch (SomeException e) {  
  
}
```

we want to write

```
public E remove4(int index) {  
    if(index < 0 || index >= size) {  
        throw new IndexOutOfBoundsException("The given index: "+index+  
            " is out of bounds for a list of size" + size);  
    }  
}
```

We are in this case detecting the problem and reporting to another part of the code (or to a developer who didn't expect this!) that they should avoid this problem. Your future self will thank you for writing a detailed error message!

On a quiz, I would just write "OOB" instead of the full message above.

⑪ See the comments about your future self thanking you under ⑩.

⑫

⑬

⑭

⑮

⑯

⑰

⑱

⑲