Question 1:

Consider the following Java method that is intended to swap two items in a `ListADT` *instance.*

```
1       void swap(int i, int j, ListADT<String> myList) {
2           String temp = myList.get(i);
3           myList.add(i, myList.get(j));
4           myList.add(j, temp);
5       }
```

For example, if `myList` *is* `["a", "b", "c", "d", "e", "f"]`, *then* `swap(1, 4, myList)` *should change* `myList` *into* `["a", "e", "c", "d", "b", "f"]`. *However, the method doesn't work as intended.*

Part A: Trace the code *on the example above by showing the contents of* `myList` *after each line of the* `swap` *method is executed.*

Part B: Rewrite the `swap` ***method*** *so that is behaves as described. You must leave the method signature unchanged. Your method must* ***not*** *return a new list; it must operate on the list instance passed in the* `myList` *parameter. You do not need to worry about input validation: you may assume that* `i` *and* `j` *are valid indices, that* `i` < `j`, *and that* `myList` *is not null.*

A.

If we're calling `swap(1, 4, myList)` and myList is `["a", "b", "c", "d", "e", "f"]`:

- Line 2: `myList` is unaltered. `temp` is instantiated with the value at myList's inde 1, which is "b", since the first index is 0. This will come into play later.
  - `["a", "b", "c", "d", "e", "f"]`
- Line 3: The value at index *j* (4) is of `myList` is added at index *i* (1). The item originally in index *i* is moves up once; the item originally in index *i+1* moves up once… this continues for each item up to the item originally in index `myList.size()-1`. `myList[4]` is `"e"`, so that's added at `myList[1]`, and everything else scoots up:
  - `["a", "e", "b", "c", "d", "e", "f"]`
- Line 4: The value of `temp`, which we established was "b", is added at position 4, scooting everything else up as described above.
  - `["a", "e", "b", "c", "b", "d", "e", "f"]`
- The general assumption that seems to have caused this to go wrong is that the *add* method would replace the item instead of inserting it.

B.

```
//Assuming i < j per the instructions (could add if i > j then switch them)

void swap(int i, int j, ListADT<String> myList) {
    String newi = myList.remove(j);
    String newj = myList.remove(i);
    myList.add(i,newi);
    myList.add(j,newj);
}
```

Question 2:

Assume that `ArrayList` implements `ListADT`, that the `ArrayList`and `ArrayListIterator` classes are implemented as expected, and that `BadListException` is an unchecked exception with a zero-argument constructor. Assume also that `null` elements may not be added to a list.

Complete the Java method specified below, making use of iterators. In order to receive full credit, your solution:

- must explicitly use iterators for traversing lists (i.e., you may not use a `for`-loop or Java's extended-`for` loop),
- must ***not*** use the `contains` method,
- may use any `ListADT` methods (except `contains`) described in the on-line reading, including `ListADT.iterator()`, but must not use any other `List` methods not mentioned there, and
- must ***not*** modify the contents of the parameters.

```
public static ListADT<String> union(ListADT<String> list1, ListADT<String> list2) {
	if (list1 == null || list2 == null) {
		throw new BadListException();
	}
	//first, combine the lists, starting with list1
	ListADT<String> combinedList = list1;
	Iterator<String> iter2 = list2.iterator();
	while (iter2.hasNext()) {
		combinedList.add(iter2.next());
	}
	//then iterate through the combined list and add non-duplicates to a new list
	Iterator<String> combinedIter = combinedList.iterator();
	ListADT<String> unionList = new ArrayList<String>();
	while (combinedIter.hasNext()) {
		String itemToCompare = combinedIter.next();
		boolean hasDuplicate = false;
		Iterator<String> iterUnion = unionList.iterator();
		while (iterUnion.hasNext()) {
			if (itemToCompare.equals(iterUnion.next())) {
				hasDuplicate = true;
			}
		}
		//if it doesn't exist in the new union list already, add it
		if (!hasDuplicate) {
			unionList.add(itemToCompare);
		}
	}
	//return the union list
	return unionList;
}
```