1. Consider a static method called `collapse` that takes an `ArrayList` of `Strings` as a parameter and that collapses successive pairs into a single `String`. Each pair should be collapsed into a new `String` that has the two values inside parentheses and separated by a comma. For example, if a variable called `list` initially stores these values:

```
["four", "score", "and", "seven", "years", "ago"]
```

and we make the following call:

```
collapse(list);
```

Your method should collapse the first pair into `"(four, score)"`, the second pair into `"(and, seven)"` and the third pair into `"(years, ago)"` so that `list` stores the following values:

```
["(four, score)", "(and, seven)", "(years, ago)"]
```

Notice that the list goes from having 6 elements to having 3 elements because each successive pair is collapsed into a single value. If there are an odd number of values in the list, the final element is not collapsed. For example, if the original list had been:

```
["to", "be", "or", "not", "to", "be", "hamlet"]
```

It would again collapse pairs of values, but the final value (`"hamlet"`) would not be collapsed, yielding this list:

```
["(to, be)", "(or, not)", "(to, be)", "hamlet"]
```

Below is a proposed implementation of `collapse`:

```
public static void collapse(ArrayList<String> list) {
   for (int i = 0; i < list.size(); i++) {
      String first = list.get(i);
      String second = list.get(i + 1);
      String pair = "(" + first + ", " + second + ")";
      list.set(i, pair);
   }
}
```

This implementation has one or more bugs. Rewrite the implementation so that it behaves as described above. Your rewritten method should retain the same basic approach to the problem as the buggy implementation.

2. Consider a static method called `expand` that takes an `ArrayList` of integer values as a parameter and that replaces pairs of integer values with sequences of numbers. In particular, the `ArrayList` will contain pairs of numbers that indicate the number of occurrences of a particular value (in that order). For example, suppose a variable called `list` stores this sequence of values:

```
[2, 4, 3, 5, 1, 9, 12, 8]
```

The first pair indicates that we want 2 occurrences of the number 4. The second pair indicates that we want 3 occurrences of the number 5. The third pair indicates that we want 1 occurrence of the number 9. The final pair indicates that we want 12 occurrences of the number 8. If we make the call:

```
expand(list);
```

We expect that `list` will store these values afterwards:

```
[4, 4, 5, 5, 5, 9, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8, 8]
```

You may assume that the `ArrayList` stores a legal sequence of values, which means that it has an even length and all of the counts are greater than or equal to 1.

Below is a proposed implementation of `expand`:

```
public static void expand(ArrayList<String> list) {
   int i = 0;
   while (i < list.size()) {
      int num = list.remove(i);
      int repeat = list.get(i);
      for (int j = 0; j < num; j++) {
         list.add(i, repeat);
      }
      i++;
   }
}
```

This implementation has one or more bugs. Rewrite the implementation so that it behaves as described above. Your rewritten method should retain the same basic approach to the problem as the buggy implementation.

1.

```
public static void collapse(ArrayList<String> list) {
   for (int i = 0; i < list.size() - 1; i++) {
      String first = list.get(i);
      String second = list.remove(i+1);
      String pair = "(" + first + ", " + second + ")";
      list.set(i, pair);
   }
}
```

2.

```
public static void expand(ArrayList<Integer> list) {
   int i = 0;
   while (i < list.size()) {
      int num = list.remove(i);
      int repeat = list.get(i);
      for (int j = 0; j < num - 1; j++) {
         list.add(i, repeat);
      }
      i += num;
   }
}
```

```
public static void expand(ArrayList<Integer> list) {
   int i = 0;
   while (i < list.size()) {
      int num = list.remove(i);
      int repeat = list.remove(i);
      for (int j = 0; j < num; j++) {
         list.add(i, repeat);
      }
      i += num;
   }
}
```