Even Odd

Here is a video walkthrough of the solutions.

Implement the method `evenOdd` by *destructively* changing the ordering of a given `IntList` so that even indexed links **precede** odd indexed links.

For instance, if `lst` is defined as `IntList.list(0, 3, 1, 4, 2, 5)`, `evenOdd(lst)` would modify `lst` to be `IntList.list(0, 1, 2, 3, 4, 5)`.

You may not need all the lines.

Hint: Make sure your solution works for lists of odd and even lengths.

```
public class IntList {
    public int first;
    public IntList rest;
    public IntList (int f, IntList r) {
        this.first = f;
        this.rest = r;
    }

    public static void evenOdd(IntList lst) {

      if (_____________________________________) {
          return;
      }

      ___________________________________________

      ___________________________________________

      while (_____________________________________) {

          ___________________________________________

          ___________________________________________

          ___________________________________________

          ___________________________________________
      }

      ___________________________________________
    }
}
```

Solution:

```
public static void evenOdd(IntList lst) {
    if (lst == null || lst.rest == null) {
        return;
    }
    IntList oddList = lst.rest;
    IntList second = lst.rest;
    while (lst.rest != null && oddList.rest != null) {
        lst.rest = lst.rest.rest;
        oddList.rest = oddList.rest.rest;
        lst = lst.rest;
        oddList = oddList.rest;
    }
    lst.rest = second;
}
```

Alternate Solution:

```
public static void evenOdd(IntList lst) {
    if (lst == null || lst.rest == null || lst.rest.rest == null) {
        return;
    }
    IntList second = lst.rest;
    int index = 0;
    while (!(index % 2 == 0 && (lst.rest == null || lst.rest.rest == null))) {
        IntList temp = lst.rest;
        lst.rest = lst.rest.rest;
        lst = temp;
        index++;
    }
    lst.rest = second;
}
```

Explanation: For any linked list, observe that we simply want to change the `rest` attribute of each `IntList` instance to skip an `IntList` instance. Looking at `lst`, we want to link 0 to 1, 3 to 4, and so on. This will constitute the work of the body of the **`while`** loop, so we just to need to figure out how to link the last even indexed `IntList` instance to the first odd indexed `IntList` instance. To keep track of the first odd indexed `IntList` instance, we can use `second`. Now, we just need to exit the **`while`** loop when we are at the last even indexed `IntList` instance. This occurs when the index is even and we are either at the second to last element (`lst.rest.rest == null`) or the last element (`lst.rest == null`).