1 Flatten

Write a method `flatten` that takes in a 2-D array `x` and returns a 1-D array that contains all of the arrays in `x` concatenated together.

For example, `flatten([{{1, 2, 3}, {}}, {7, 8}])` should return `[1, 2, 3, 7, 8]`.

(Summer 2016 MT1)

```java
public static int[] flatten(int[][] x) {
    int totalLength = 0;
    for (int i = 0; i < x.length; i++) {
        totalLength += x[i].length;
    }
    int[] a = new int[totalLength];
    int aIndex = 0;
    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {
            a[aIndex] = x[i][j];
            aIndex++;
        }
    }
    return a;
}
```
Solution:

```java
public static int[] flatten(int[][] x) {
    int totalLength = 0;
    for (int[] arr: x) {
        totalLength += arr.length;
    }
    int[] a = new int[totalLength];
    int aIndex = 0;
    for (int[] arr: x) {
        for (int value: arr) {
            a[aIndex] = value;
            aIndex++;
        }
    }
    return a;
}
```

Alternate Solutions:

```java
public static int[] flatten(int[][] x) {
    int totalLength = 0;
    for (int[] arr: x) {
        totalLength += arr.length;
    }
    int[] a = new int[totalLength];
    int aIndex = 0;
    for (int[] arr: x) {
        System.arraycopy(arr, 0, a, aIndex, arr.length);
        aIndex += arr.length;
    }
    return a;
}

public static int[] flatten(int[][] x) {
    int totalLength = 0;
    for (int i = 0; i < x.length; i++) {
        totalLength += x[i].length;
    }
    int[] a = new int[totalLength];
    int aIndex = 0;
    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {
            a[aIndex] = x[i][j];
            aIndex++;
        }
    }
    return a;
}
```
2 Skippify

Suppose we have the following IntList class, as defined in lecture and lab, with an added skippify function.

Suppose that we define two IntLists as follows.

```java
IntList A = IntList.list(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);
IntList B = IntList.list(9, 8, 7, 6, 5, 4, 3, 2, 1);
```

Fill in the method skippify such that the result of calling skippify on A and B are as below:
- After calling A.skippify(), A: (1, 3, 6, 10)
- After calling B.skippify(), B: (9, 7, 4)

(Spring '17, MT1)

```java
public class IntList {
    public int first;
    public IntList rest;

    @Override
    public boolean equals(Object o) { ... }
    public static IntList list(int... args) { ... }

    public void skippify() {
        IntList p = this;
        int n = 1;
        while (p != null) {
            IntList next = __________________________;

            for (_______________________________________) {
                if (________________________________________) {
                    _________________________________
                }
            }
        }
    }
}
```
Solution:

```java
public class IntList {
    public int first;
    public IntList rest;

    @Override
    public boolean equals(Object o) { ... }
    public static IntList list(int... args) { ... }

    public void skippify() {
        IntList p = this;
        int n = 1;
        while (p != null) {
            IntList next = p.rest;
            for (int i = 0; i < n; i += 1) {
                if (next == null) {
                    break;
                }
            }
            next = next.rest;
            p = p.rest;
            n++;
        }
    }
}
```

**Explanation:** Looking at `IntList A`, we only need to change the `rest` attribute of `IntList` instances 1, 3, and 6. To achieve this, we will use the `for` loop to find the new `rest` attribute (which we will store in `next`) of the current `IntList` instance (`p`). The outer `while` loop enables us to repeat these actions for, in our case, `IntList` instances 3 and 6. The `int n` will increment by one each iteration and gives us the number of iterations in the for loop, i.e. how many `IntList` instances to skip. Finally, the `if` check accounts allows us to exit the for loop early if we ever hit the end of the Linked List.
3 Even Odd

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).

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

```java
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;
        }

        IntList second = __________________;

        int index = _______________________;  

        while (____________________________________________) {
            __________________________________________
            __________________________________________
            __________________________________________
            __________________________________________
            }

        __________________________________________
    }
}
```
Solution:

```java
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 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).

Alternate Solution:

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