Flatten

Here is a video walkthrough of the solutions.

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}`.

```
public static int[] flatten(int[][] x) {
    int totalLength = 0;

    for (_____________________________________) {

        ___________________________________________
    }

    int[] a = new int[totalLength];
    int aIndex = 0;
    for (_____________________________________) {

        ___________________________________________

        ___________________________________________

        ___________________________________________

        ___________________________________________
    }

    return a;
}
```

Solution:

```
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:

```
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 = newint[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;
}
```

Here is a video walkthrough of the solutions for this problem.

Explanation: All these solutions do essentially the same thing. In Java, an array's length must be known before we can instantiate it–as such, we have to loop over all inner arrays to get the `totalLength` of our flattened array. Then, we iterate over the elements of `x`, filling `a` as we go. `aIndex` keeps track of where we are in the `a` array.