When you call a method and the method has an array as a parameter, the array you pass is not copied. In fact, all that is copied is the address in memory of where the array starts. This way of passing arrays is named"call-by-reference" because just a reference is passed. It is more efficient than would be passing a copy of the array, especially for large arrays.

The big consequence of this is that a method can manipulate the passed-in array.

```
int x = 0;
foo(x);
// x still 0 no matter what foo does.

int[] arr = new int[5];         // new always zeros its creation
bar(arr);
// arr could be changed by bar
```

1) Write a static method called incrementAll that takes an integer array as a parameter and increases each of its elements by 1.

   ```
   public static void incrementAll(int[] arr) {
   ```

Sometime this behavior is desired. For example Arrays.sort(arr) rearranges the elements in arr in increasing order and that's exactly the reason it exists. But unless a method is documented as changing the contents of an array that's passed-in, it's considered bad practice for it to do so.

2) Write a static method called median that takes an array of doubles as a parameter and returns the median element. Do not alter the array that's been passed in. Investigate the method copyOf in the Arrays class and use it.

3) Write a static method called reverse that takes an array of ints as a parameter and returns a new array that is the same as the original array but in reverse order. Do not alter the array that's been passed in.

4) Write a static method called reverseInPlace that takes an array of ints as a parameter and reverses its order. Do not create an extra array.

If you have not already, and if time permits, implement and test some of your solutions to the problems above.