1 Read Me

Describe what each of the following methods does. You may assume that `values` contains at least one element.

```java
private static boolean method1 (int[] values) {
    int k = 0;
    while (k < values.length - 1) {
        if (values[k] > values[k+1]) {
            return false;
        }
        k = k + 1;
    }
    return true;
}
```

**Solution:** `method1` returns true if `values` is non-decreasing, i.e. if each value in `values` is larger than or equal to the previous element.

```java
private static void method2 (int[] values) {
    int k = 0;
    while (k < values.length / 2) {
        int temp = values[k];
        values[k] = values[values.length - 1 - k];
        values[values.length - 1 - k] = temp;
        k = k + 1;
    }
}
```

**Solution:** `method2` reverses `values` in place. Note that `method2` has no return value and instead mutates `values`. 
2 CopyCat

For the following class, write a non-static method called cloneCat that allows the current Cat to clone itself. (Hint: This means incrementing the clones field and returning a clone of the current Cat object using the provided constructor.)

```java
class Cat {
    public static int clones = 5;
    String name;

    public Cat() {
        name = "Catherine";
    }

    public Cat(Cat c) {
        name = c.name;
    }

    public Cat cloneCat() {
        clones++;  
        return new Cat(this);
    }
}
```

Could you call cloneCat from an instance object? How about from a class?

**Solution:** We can create an instance object and call the method.

```java
Cat c = new Cat(); 
c.cloneCat();
```

No, we cannot call the method from a class. The method is non-static and attempting to run the code would yield a compile-time error.

What would happen if we added the static keyword to cloneCat without modifying the body of the method? If we changed the method body as well, how could we call cloneCat from the class? Would we be able to call cloneCat from an instance object?

**Solution:**

If we added the static keyword, the method would not make sense. There would be a compile-time error from referencing a non-static variable from a static context. This is because the keyword this refers to the instance of Cat that calls cloneCat. Since we are making cloneCat static, and therefore making it independent of the instances of cloneCat, this no longer makes any sense.
However, if the method body was changed so that it did work, we would call it with:

```java
Cat.cloneCat();
```

Yes, it is okay to call static methods from an instance object.
3 Flatten

Write a method flatten that takes in a 2-D int array x and returns a 1-D int array that contains all of the arrays in x concatenated together. For example, flatten([{{1, 3, 7}}, {}, {9}]) should return [1, 3, 7, 9].

Solution:

```java
public static int[] flatten(int[][] x) {
    //newSize will hold the length of the flattened list
    int newSize = 0;

    for (int i = 0; i < x.length; i++) {
        //calculating the length of flattened list
        newSize += x[i].length;
    }

    int[] toReturn = new int[newSize];

    //toReturnIndex will be the index used to access the flattened list
    int toReturnIndex = 0;

    for (int i = 0; i < x.length; i++) {
        for (int j = 0; j < x[i].length; j++) {

            /* index into the flattened list using toReturnIndex
            and store the element from the original
            2D-array at position (i, j) */

            toReturn[toReturnIndex] = x[i][j];

            /* increment the toReturnIndex for next time
            (next position in the flattened array) */

            toReturnIndex += 1;
        }
    }

    return toReturn;
}
```