CS 61B
Summer 2021

Small Group Tutoring
Section 2: Arrays [Solution]

Worksheet 2

1 Read Me

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

```
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/in ascending order. In other words, it checks if each value in `values` is larger than *or equal to* the previous element.

```
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 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:

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

    for (int i = 0; i < x.length; i+=1) {
        //calculating the length of flattened list
        newArraySize += x[i].length;
    }
    int[] newArray = new int[newArraySize];

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

    for (int i = 0; i < x.length; i+=1) {
        for (int j = 0; j < x[i].length; j+=1) {

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

            newArray[newArrayIndex] = x[i][j];

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

            newArrayIndex += 1;
        }
    }
    return newArray;
}
```

3 Bugged Out

We have a class arrFunctions, and we decide that we want to write it a method with the following signature: `public static int arr_multiply(int[] arr)`. This method takes in an `int[]` and returns all the **non-zero** values in the array multiplied together. If there is a zero in the array, we want to ignore it. The only time we should return 0 is if the array is empty. We want our array to work in all sorts of odd edge cases without any errors.

Write 3 unit tests that each target cases for this method. You do not need to write the method, just the tests (don't you love test driven development?!).

We have written 4 tests below. Other good test ideas might have been negative numbers or extremely large arrays.

```
@Test
public void testRegular() {
    int[] arr = new int[]{1, 2, 3}
    int result = arrFunctions.arr_multiply(arr);
    assertEquals(result, 6);
}

@Test
public void testZero() {
    int[] arr = new int[]{0, 2, 3}
    int result = arrFunctions.arr_multiply(arr);
    assertEquals(result, 6);
}

@Test
public void testEmpty() {
    int[] arr = new int[]{}
    int result = arrFunctions.arr_multiply([]);
    assertEquals(result, 0);
}

@Test
public void testOne() {
    int[] arr = new int[]{61}
    int result = arrFunctions.arr_multiply(arr);
    assertEquals(result, 61);
}
```

4 Extra: Static Electricity

```
public class Pokemon {
    public String name;
    public int level;
    public static String trainer = "Ash";
    public static int partySize = 0;

    public Pokemon(String name, int level) {
        this.name = name;
        this.level = level;
        this.partySize += 1;
    }

    public static void main(String[] args) {
        Pokemon p = new Pokemon("Pikachu", 17);
        Pokemon j = new Pokemon("Jolteon", 99);
        System.out.println("Party size: " + Pokemon.partySize);
        p.printStats();
        int level = 18;
        Pokemon.change(p, level);
        p.printStats();
        Pokemon.trainer = "Ash";
        j.trainer = "Brock";
        p.printStats();
    }

    public static void change(Pokemon poke, int level) {
        poke.level = level;
        level = 50;
        poke = new Pokemon("Voltorb", 1);
        poke.trainer = "Team Rocket";
    }

    public void printStats() {
        System.out.println(name + " " + level + " " + trainer);
    }

}
```

a) Write what would be printed after the main method is executed.

```
Party Size: 2
Pikachu 17 Ash
Pikachu 18 Team Rocket
Pikachu 18 Brock
```

b) On line 28, we set `level` equal to `50`. What `level` do we mean? An instance variable of the `Pokemon` object? The local variable containing the parameter to the `change` method? The local variable in the `main` method? Something else?

It is the local variable in the `change` method and does not have any effect on the other variables of the same name in the `Pokemon` class or the `main` method.

c) If we were to call `Pokemon.printStats()` at the end of our main method, what would happen?

If we were to add this line to our main method, it would error. In the class, `printStats()` is an instance method. What would it mean to print the name and level of the class `Pokemon`, as opposed to a specific Pokemon's name? It doesn't really make sense. So when we try to run this method on our class, it errors.

One more thing to note is the method `change` is declared static itself. Static methods can be called using the name of the class, as in line 19, whereas non-static methods cannot. The golden rule for static methods to know is that **static methods can only modify static variables**.