/trasteando/

Playing around

-
-
-
- in

Search...

Navigate...

- Blog
- Archives
- Readings

Are They the Same? Kata Java Solution

Sep 2nd, 2015 10:47 pm

Today I was practicing with the Are they the same? kata from www.codewars.com

Here's a summary of the steps I followed to solve it.

Description

The kata goal is

> Given two arrays a and b write a function comp(a, b) (compSame(a, b) in Clojure) that checks whether the two arrays have the "same" elements, with the same multiplicities. "Same" means, here, that the elements in b are the elements in a squared, regardless of the order.

The full description can be found at www.codewars.com

The initial code

The initial code has one class and a test.

```
1 public class AreSame {
2
3   public static boolean comp(int[] a, int[] b) {
4     return null;
```

```
5   }
6 }
```

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4
5  public class AreSameTest {
6
7    @Test
8    public void test1() {
9        int[] a = new int[]{121, 144, 19, 161, 19, 144, 19, 11};
10       int[] b = new int[]{121, 14641, 20736, 361, 25921, 361, 20736, 361};
11       assertEquals(AreSame.comp(a, b), true);
12   }
13 }
```

The kata focuses on implementing the `AreSame` class so it passes the tests provided by the `AreSameTest` class.

It also seems a good idea to enhance the tests in `AreSameTest` to further test our implementation.

Solution steps

Iteration 1 - true use case

My goal for this first iteration was to make `test1` pass / green.

The `comp` method will

- sort array `a`
- sort array `b`
- raise `a` elements to the second power
- compare the elements in `a` and `b`

This is the resulting code:

```
1  public static boolean comp(int[] a, int[] b) {
2      Arrays.sort(a);
3      Arrays.sort(b);
4      for (int i = 0; i < a.length; i++) {
5          if (a[i]*a[i] != b[i]) {
6              return false;
7          }
8      }
9      return true;
10 }
```

This code has a couple of problems that will be dealt with in next iterations:

- `Arrays.sort()` is modifying the input parameters, which can be considered as a code smell.
- The `for` loop used to compare both arrays adds a lot of code and is not expressive enough, I'm sure there are better ways to achieve the same result

Iteration 2 - false use case

The tests provided are only covering the scenario where `comp` return `true`. Let's add a test for `comp` returning false.

```
1 @Test
2 public void test2() {
3     int[] a = new int[]{122, 144, 19, 161, 19, 144, 19, 11};
4     int[] b = new int[]{121, 14641, 20736, 361, 25921, 361, 20736, 361};
5     assertEquals(AreSame.comp(a, b), false);
6 }
```

The new test is also satisfied by the current `comp` implementation, so there's no need to modify the code at this stage.

Iteration 3 - null / empty use case

The kata description mentions that we have to deal with null and empty arrays:

> If a or b are nil (or null or None), the problem doesn't make sense so return false.
>
> If a or b are empty the result is evident by itself.

Let's add two new tests to cover for these scenarios:

```
1  @Test
2  public void testNull() {
3      assertEquals(AreSame.comp(null, null), false);
4  }
5
6  @Test
7  public void testEmpty() {
8      int[] a = new int[]{};
9      int[] b = new int[]{};
10     assertEquals(AreSame.comp(a, b), true);
11 }
```

When running the test suite, `testEmpty` is passing, but `testNull` is failing: `comp` needs to be modified to deal properly with null arrays an to bring the test suite back to green.

A check for null arguments is added to the `comp` method to turn the tests back to pass / green.

```
1 if (a == null || b == null) return false;
```

and the resulting `comp` method is as follows:

```
1  public static boolean comp(int[] a, int[] b) {
2      if (a == null || b == null) return false;
3
4      Arrays.sort(a);
5      Arrays.sort(b);
6      for (int i = 0; i < a.length; i++) {
7          if (a[i]*a[i] != b[i]) {
```

```
8               return false;
9           }
10      }
11      return true;
12  }
```

Iteration 4 - Different sizes use case

How will `comp` deal with arrays of different sizes? Let's add a couple of tests to check this scenarios:

```
1  @Test
2  public void testBLonger() {
3      int[] a = new int[]{121, 144, 19, 161, 19, 144, 19, 11};
4      int[] b = new int[]{121, 14641, 20736, 361, 25921, 361, 20736, 361, 2073600};
5      assertEquals(AreSame.comp(a, b), false);
6  }
7
8  @Test
9  public void testALonger() {
10     int[] a = new int[]{121, 144, 19, 161, 19, 144, 19, 11, 14400};
11     int[] b = new int[]{121, 14641, 20736, 361, 25921, 361, 20736, 361};
12     assertEquals(AreSame.comp(a, b), false);
13 }
```

Just by adding a couple of big numbers to the end of `a` and `b` we are making our code fail.

In order to bring the tests back to pass / green a guard to check the size of the arrays can be added:

```
1 if (a.length != b.length) return false;
```

and the resulting `comp` method is as follows:

```
1  public static boolean comp(int[] a, int[] b) {
2
3      if (a == null || b == null) return false;
4      if (a.length != b.length) return false;
5
6      Arrays.sort(a);
7      Arrays.sort(b);
8      for (int i = 0; i < a.length; i++) {
9          if (a[i]*a[i] != b[i]) {
10             return false;
11         }
12     }
13     return true;
14 }
```

Iteration 5 - Negative numbers use case

There's a scenario that hasn't been tested yet: what if `a` contains negative numbers?

A new test is added to cover for this scenario:

```
1 @Test
2 public void testNegativeNumbers() {
3     int[] a = new int[]{121, -144, 19, 161, 19, 144, 19, 11};
4     int[] b = new int[]{121, 14641, 20736, 361, 25921, 361, 20736, 361};
5     assertEquals(AreSame.comp(a, b), true);
6 }
```

When the tests are run we see this one failing miserably: the code needs to be changed to bring the test suite back to green.

The error is caused by the fact that `comp` sorts `a` and then raises its values to the second power, which doesn't work well for negative numbers.

At a first stage I considered working with absolute values from the `a` array. In order to apply `Math.abs()` to the values in the `a` array it seemed inevitable to add a new `for` loop, but I felt lazy so I after googling a bit I found that `lambdas` may come in to rescue.

The following code snippet returns a new array with the `abs` function applied to all its elements:

```
1 Arrays.stream(a).map(x -> Math.abs(x)).toArray();
```

Arrays can be also sorted by calling:

```
1 Arrays.stream(b).sorted().toArray();
```

Here's the resulting `comp` function after making use of `Arrays`, `IntStreams` and `lambdas`:

```
1  import java.util.Arrays;
2
3  public class AreSame {
4
5      public static boolean comp(int[] a, int[] b) {
6
7          if (a == null || b == null) return false;
8          if (a.length != b.length) return false;
9
10         int[] sortedA = Arrays.stream(a).map(Math::abs).sorted().toArray();
11         int[] sortedB = Arrays.stream(b).sorted().toArray();
12         for (int i = 0; i < a.length; i++) {
13             if (sortedA[i]*sortedA[i] != sortedB[i]) {
14                 return false;
15             }
16         }
17         return true;
18     }
19 }
```

And that brought the tests back to pass / green. Now it's time for some refactoring!

Iteration 5 - Refactoring

By using `IStreams` the original arrays passed in to the `comp` method remain untouched so we have already dealt with one of the concerns raised in our first iteration.

Let's see how the `comp` method can be refactored, and if we can get rid of the `for` loop in it used for array comparison.

Now that the `comp` method uses `lambdas` to apply a function to all the elements of an array, why not use `lambdas` to square the elements of `a`? By doing so `Arrays.equals` could be used to compare the resulting array with `b` and save our code for all the overhead in the `for` loop used for comparison.

The following code snippet applies `abs`, squares and sorts the elements in the `a` array:

```
1 Arrays.stream(a).map(x -> Math.abs(x)).map(x -> x*x).sorted().toArray();
```

Wait a minute! If we raise to the second power, `abs` is redundant, so we can get rid of it:

```
1 Arrays.stream(a).map(x -> x*x).sorted().toArray();
```

And putting it all together here is the `comp` method final version:

```
1 public static boolean comp(int[] a, int[] b) {
2
3     if (a == null || b == null) return false;
4     if (a.length != b.length) return false;
5
6     int[] sortedA = Arrays.stream(a).map(x -> x*x).sorted().toArray();
7     int[] sortedB = Arrays.stream(b).sorted().toArray();
8     return Arrays.equals(sortedA, sortedB);
9 }
```

Here are the final versions of the classes of this kata:

- AreSame.java
- AreSameTest.java

Authored by Angel Rojo Sep 2nd, 2015 10:47 pm java, kata

« Bootstrapping an Agile Team [Part 1 – People] - by Brendan Marsh What Not To Do In a TDD Pair Programming Interview - by Jason Gorman @ Software People Inspiring »