assertIterableEquals method

This lesson demonstrates how to use assertIterableEquals method in JUnit 5 to assert test conditions.

WE'LL COVER THE FOLLOWING

assertIterableEquals() method

Assertions API provide static `assertIterableEquals()` method. This method helps us in validating that `expected` and `actual` `iterables` are deeply equal. By, deeply equal we mean that number and order of elements in the collection must be the same, as well as iterated elements must be equal.

There are basically three useful overloaded methods for assertIterableEquals:-

```
public static void assertIterableEquals(Iterable<?> expected, Iterable> actual)

public static void assertIterableEquals(Iterable<?> expected, Iterable> actual, String messag

public static void assertIterableEquals(Iterable<?> expected, Iterable> actual, Supplier<Stri
```

Demo

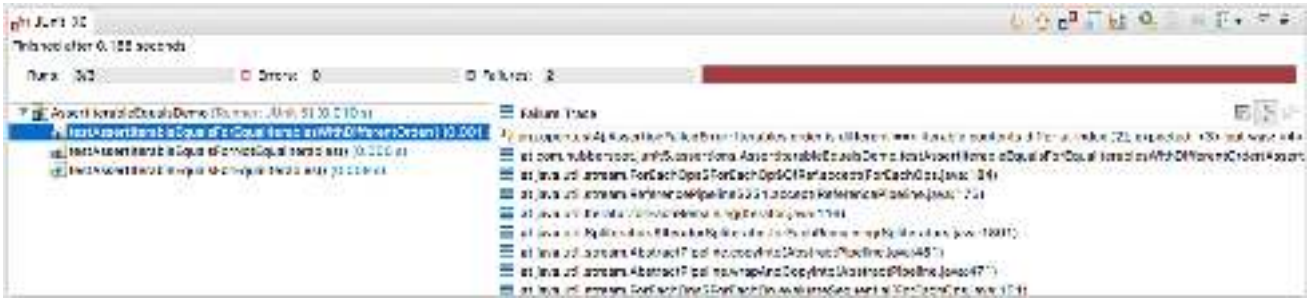
Let's look into the usage of the above methods:-

assertIterableEquals method

```
package io.educative.junit5;

import static org.junit.jupiter.api.Assertions.assertIterableEquals;

import java.util.ArrayList;
import java.util.Arrays;

import org.junit.jupiter.api.Test;

public class AssertIterableEqualsDemo {

        @Test
        public void testAssertIterableEqualsForEqualIterables() {
                  Iterable<Integer> expected = new ArrayList<>(Arrays.asList(1,2,3,4));
            Iterable<Integer> actual = new ArrayList<>(Arrays.asList(1,2,3,4));
            assertIterableEquals(expected, actual);
        }

        @Test
        public void testAssertIterableEqualsForNotEqualIterables() {
                  Iterable<Integer> expected = new ArrayList<>(Arrays.asList(1,2,3,4));
            Iterable<Integer> actual = new ArrayList<>(Arrays.asList(1,2,3));
            assertIterableEquals(expected, actual, "Iterables are not equal.");
        }

        @Test
        public void testAssertIterableEqualsForEqualIterablesWithDifferentOrder() {
                  Iterable<Integer> expected = new ArrayList<>(Arrays.asList(1,2,3,4));
            Iterable<Integer> actual = new ArrayList<>(Arrays.asList(1,2,4,3));
```

```
            assertIterableEquals(expected, actual, () -> "Iterables order is different");
        }
}
```

Run AssertIterableEqualsDemo class as JUnit Test.



Explanation -

In the AssertIterableEqualsDemo class, there are 3 @Test methods. These 3 methods demonstrate the working of the above 3 overloaded methods of `assertIterableEquals`:-

1. `testAssertIterableEqualsForEqualIterables()` - It asserts that actual and expected iterables are equal. Here, the expected iterable is ArrayList with four elements as, {1,2,3,4} and actual iterable is ArrayList with four elements as, {1,2,3,4}. Thus, it passes the Junit test case because `assertIterableEquals` finds actual and expected iterables to be equal.

2. `testAssertIterableEqualsForNotEqualIterables()` - It asserts that actual and expected iterables are equal. Here, the expected iterable is arraylist with four elements as, {1,2,3,4} and actual iterable is arraylist with four elements as, {1,2,3}. Thus, it fails the Junit test case with `AssertionFailedError:` Iterables are not equal. ==> iterable lengths differ, expected: <4> but was: <3> because `assertIterableEquals` finds actual and expected iterables not equal. It gives `AssertionFailedError` followed by `String message` we provide to assertIterableEquals() method.

3. `testAssertIterableEqualsForEqualIterablesWithDifferentOrder()` - It asserts that actual and expected iterables are equal. Here, the expected iterable is arraylist with four elements as, {1,2,3,4} and actual iterable is arraylist with four elements as, {1,2,4,3}. Thus, it fails the Junit test case with `AssertionFailedError:` Iterables order is different ==> iterable

with `AssertionFailedError:` Iterables order is different ==> Iterable contents differ at index [2], expected: <3> but was: <4> because though

contents of iterables are same they are not in same order. It gives `AssertionFailedError` followed by lazily evaluated `String message` we provide to assertIterableEquals() method, as lambda expression.

In the next lesson, we will look into `assertThrows()` assertion.