assertNotSame() method

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

We'll cover the following

assertNotSame() method

Assertions API provide static `assertNotSame()` method. This method helps us in validating that `expected` and `actual` do not refer to the exact same object. JUnit uses `==` operator to perform this assert.

- If the actual and expected value refers to the same object then the test case will fail.
- If the actual and expected value does not refer to the same object then the test case will pass.

There are basically three useful overloaded methods for assertNotSame:-

```
1  public static void assertNotSa
2
3  public static void assertNotSa
4
5  public static void assertNotSa
```

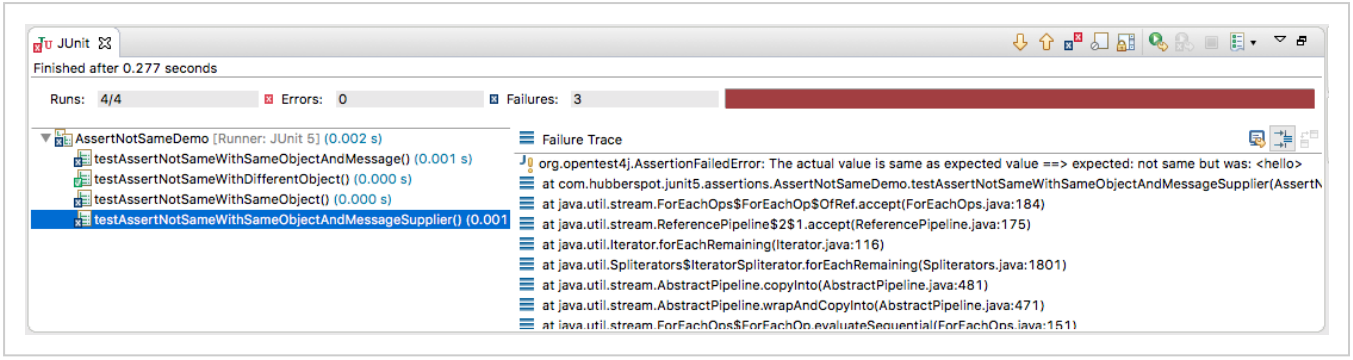
Demo

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

```
1  package io.educative.junit5;
2
3  import static org.junit.jupiter.api.Assertions.assertNotSame;
```

```java
import org.junit.jupiter.api.Test;

public class AssertNotSameDemo {

    @Test
    public void testAssertNotSameWithDifferentObject() {
        String actual = "hello";
        String expected = "hell";
        assertNotSame(expected, actual);
    }

    @Test
    public void testAssertNotSameWithSameObject() {
        String actual = "hello";
        String expected = "hello";
        assertNotSame(expected, actual);
    }

    @Test
    public void testAssertNotSameWithSameObjectAndMessage() {
        String actual = "hello";
        String expected = "hello";
        assertNotSame(expected, actual, "The actual value is same as expected value");
    }

    @Test
    public void testAssertNotSameWithSameObjectAndMessageSupplier() {
```

Run AssertNotSameDemo class as JUnit Test.



Explanation -

In the AssertNotSameDemo class, there are 4 @Test methods. These 4 methods demonstrate the working of the above 3 overloaded methods of `assertNotSame`:-

1. `testAssertNotSameWithDifferentObject` - It asserts that actual value does not refer to same expected object. Here, the expected value and actual value passed to `assertNotSame()` are `hell` and `hello`. Thus, it passes the

Junit test case because `assertNotSame` finds actual and expected objects not same.

2. `testAssertNotSameWithSameObject` - It asserts that actual value does not refer to same expected object. Here, the expected value and actual value passed to `assertNotSame()` is `hello`. Thus, it fails the Junit test case with `AssertionFailedError:` expected: not same but was: <hello> because ‘hello’ and ‘hello’ are same String objects.

3. `testAssertNotSameWithSameObjectAndMessage` - It asserts that actual value does not refer to same expected object. Here, the expected value and actual value passed to `assertNotSame()` is `hello`. Thus, it fails the Junit test case with `AssertionFailedError:` The actual value is same as expected value ==> expected: not same but was: <hello> because ‘hello’ and ‘hello’ are same String objects. It gives `AssertionFailedError` followed by `String message` we provide to assertNotSame() method.

4. `testAssertNotSameWithSameObjectAndMessageSupplier` - It asserts that actual value does not refer to same expected object. Here, the expected value and actual value passed to `assertNotSame()` is `hello`. Thus, it fails the Junit test case with `AssertionFailedError:` The actual value is same as expected value ==> expected: not same but was: <hello> because hello and hello are same String objects. It gives `AssertionFailedError` followed by lazily evaluated `String message` we provide to assertNotSame() method, as lambda expression.

In the next lesson, we will look into `assertArrayEquals()` and `assertIterableEquals()` assertions.