The third assignment

NOTE: This assignment is going to continue where we left off in the second one. It is highly recommended that the second assignment is completed prior to working on this one.

This assignment will focus on introducing Junit test runner and writing actual tests.

Laying out the strucutre of our first test class

1. Add Junit 4.12 as a dependency
    a. Junit - LINK
2. Create a new class named SmokeTest under **src > test > java**
    a. If you would like to learn more about Maven Standard Directory Layout - LINK
3. Add a method that is public, has a return type of void and is named searchResultTest()
4. Add another method that is public, has a return type of void ans is named socialMediaTest()
5. Add annotation above both methods which indicates that public void method to which it is attached can be executed as a test case
    a. Resource - LINK
6. Add a method that is public, **static**, has a return type of void and is named setup()
7. Add annotation above this method that denotes that we want it to run once **before any of the test methods** in the class
    a. Resource – LINK
8. Add another method that is public, **static**, has a return type of void and is named cleanup()
9. Add annotation above this method that denotes that we want it to run **once after all the tests in the class have been run**
    a. Resource – LINK, section JUnit Annotations
10. Add a field to the class that is private, has a return type equal to our previously created class for centralized page instantiation i.e Imdb and is named **imdb.**
11. In the body of setup() method, add code that will assign a new instance of Imdb class to the newly created imdb field.
12. In the body of cleanup() method, add code that will:
    a. Call the getBrowser() static method from our Browser class
    b. Call the quit() method from the result of getBrowser() method
13. We are now ready to start adding tests!

Adding the tests

1. In the body of the searchResultTest(), add logic that will do the following:
    - Navigate to https://www.imdb.com
    - Enter “12 Angry men” (or any other search term) in the search bar
    - Click on Search button
    - Assert that the title of the first result contains our search term

    Resources - Junit assertion, LINK

2. In the body of the socialMediaTest(), add logic that will do the following:
    - Navigate to https://www.imdb.com
    - Click on Sign in button
    - Assert multiple conditions that include the following:
        i. Assert that the first sign in method is Imdb
        ii. Assert that the second sign in method is Amazon
        iii. Assert that the third sign in method is Facebook
        iv. Assert that the fourth sign in method is Google
3. Debug all of your tests in one go
4. Fix any issues, wrong locators or logical errors that might pop up
5. Debug all of your tests again
6. Enjoy your new creation!

Slight refactoring

- **Planning phase:**
  - Realize that, if we wanted to add another test suite next to our SmokeTest e.g. Regression run, we would have to do the setup there all over again
  - Notice how the first step of every test is to navigate to imdb url
  - Vow to address this immediately!
- **Execution phase:**
  - Create another class under src > test > java named BaseTest
  - Move the imdb field to BaseTest and change its access modifier to **protected**
    - We change it to protected in order for it to be available in **any class that inherits** BaseTest
  - Move the setup() and cleanup() methods to BaseTest, together with their annotations
  - Make the SmokeTest class inherit the BaseTest class
  - Within the **SmokeTest** class, introduce a new method that is public, has a return type of void and is named testSetup()
  - Add annotation above this method that denotes this method should be run **before every method decorated with @Test**
    - Resource: LINK
  - Remove the step „Navigate to https://www.imdb.com“ from **both test methods**
  - Add this step to the body of the previously added testSetup() method
  - Run the tests again and verify that they worked exactly as before the refactoring

Now we would be able to add a separate test suite and all we would require for the almost complete setup of tests is to inherit the BaseTest class and we would be ready to go!

Introducing an optional test

NOTE: This section is completely **optional**. Even though the steps of this test may seem simple, the implementation will heavily rely on the abstract concepts behind our framework. It will include extending our existing framework in a certain way. To make it an actual test of understanding, the steps for implementing it will be minimal, with only several tips provided. Do not worry if you are not able to complete it.

1. Add a method to SmokeTest class named top250Test()
2. The logic of the tests should be to:
    a. Navigate to imdb
    b. Navigate to Top Rated Movies page
    c. Assert that there are **exactly 250** movies in the list
    d. Assert that the **top** rated movie is **The Shawshank Redemption**
    e. Assert that the **second** rated movie is from **1972**
    f. Assert that there **are exactly two movies with 9.0 rating**
    g. Note that the exact counts may change at the time of writing the test

Resources:

- Finding multiple elements - LINK
- Getting the number of members of a list - LINK
- Xpath suggestion - LINK, section “Other things”, Text match

Completed optional test?

Continue adding tests that you believe should cover important functions. Change the website under test. Create a new test suite class. Practice, practice, practice! The only way to master a skill is to practice, just like drawing, for example...

Resource - LINK