COMS/SE 319: Construction of User Interfaces
Spring 2021

Lab Activity 5 – Unit Testing using Jest framework & UI Testing using Selenium

Part I: Unit Testing Using Jest Framework

1. Installing Jest

- Download Lab05-TestingPlatforms-SampleCodes.zip and unzip the folder to your computer.
- Open your command prompt and navigate to the folder "Lab05-TestingPlatforms-SampleCodes/Jest-Codes"
- Run the command **npm install jest**

```
C:\Users\jsusan\Documents\Spring 2020\COMS319\Week 8\Lab07_Code>npm install jest
npm WARN deprecated request@2.88.2: request has been deprecated, see https://github.com/requ
npm notice created a lockfile as package-lock.json. You should commit this file.
npm WARN optional SKIPPING OPTIONAL DEPENDENCY: fsevents@^2.1.2 (node_modules\jest-haste-map
npm WARN notsup SKIPPING OPTIONAL DEPENDENCY: Unsupported platform for fsevents@2.1.2: wante
"} (current: {"os":"win32","arch":"x64"})
npm WARN Lab07_Code@1.0.0 No description
npm WARN Lab07_Code@1.0.0 No repository field.

+ jest@25.1.0
added 486 packages from 285 contributors and audited 1203821 packages in 20.918s

21 packages are looking for funding
  run `npm fund` for details

found 0 vulnerabilities
```

The npm modules will be installed under Lab06-TestingPlatforms-SampleCodes/Jest-Codes folder.

2. Running Sample Code

- Run the command **npm init -y**

```
C:\Users\jsusan\Documents\Spring 2020\COMS319\Week 8\Lab07_Code>npm init -y
Wrote to C:\Users\jsusan\Documents\Spring 2020\COMS319\Week 8\Lab07_Code\package.json:

{
  "name": "Lab07_Code",
  "version": "1.0.0",
  "description": "",
  "main": "callback.test.js",
  "dependencies": {
    "jest": "^25.1.0"
  },
  "devDependencies": {},
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "keywords": [],
  "author": "",
  "license": "ISC"
}
```

This creates a file called package.json in the "Lab06-TestingPlatforms-SampleCodes/Jest-Codes" folder. If the installation is successful, you must see "jest" under dependencies as shown in yellow box above.

- Modify the “test” value to be “jest” in package.json as shown below

```
"scripts": {
    "test": "jest"
  },
```

- Run the command **npm run test**

```
C:\Users\jsusan\Documents\Spring 2020\COMS319\Week 8\Lab07_Code>npm run test

> Lab07_Code@1.0.0 test C:\Users\jsusan\Documents\Spring 2020\COMS319\Week 8\Lab07_Code
> jest

 PASS  ./callback.test.js
  √ the data is peanut butter (1ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        1.333s
Ran all test suites.
```

- **CHECKPOINT:** Make sure all your tests pass, and see an output similar to shown above.

Reference: https://jestjs.io/docs/en/getting-started

2.Understanding the Sample Code

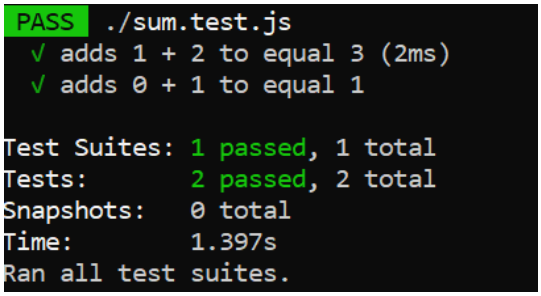
1. Adding a new test case

Testing function **sum(a, b)** defined in sum.js

To our test suite **sum.test.js**, add another test case to test if our sum function correctly adds two numbers 0 and 1.

```
test('adds 0 + 1 to equal 1', () => {
  expect(sum(0, 1)).toBe(0);
});
```

- **CHECKPOINT:** If you copy pasted the code above, and ran the command **npm run test** one of the tests have failed. Correct it and run again, until your test pass.



2. Matchers

Every test case has an actual value evaluated by the function that we are testing, and the expected value. So, we need a mechanism that would validate the expected output with the actual output.

Read the code in *matchers.test.js*
https://jestjs.io/docs/en/using-matchers
https://jestjs.io/docs/en/expect

3. Mocks

Understand how to create mock functions and mock return values.

Read the code in *mock.test.js*
https://jestjs.io/docs/en/mock-functions

Outside Lab Reading: Our test cases may depend on fetching a large file from a server. But we may want to run our tests very fast, in those cases we would want to override the original implementation of "fetching large file from a server" to a mock one. Jest allows us to create manual mocks. For more information read https://jestjs.io/docs/en/manual-mocks

4. Setup and Teardown

Understand how to specify a setup function to be called before a test case. This can be used to setup some object instances or function calls that are necessary for the tests to run. In the teardown phase, the resources created during setup are destroyed or closed.

Read the code in *settear_all.test.js* and *settear_each.test.js*
https://jestjs.io/docs/en/setup-teardown

- **CHECKPOINT:** Do you understand how to write some simple test cases for your code? Are you familiar with some common matchers you can use to compare your expected test output with the actual one evaluated by your code?

Part II: Web UI Testing with Selenium

What is GUI Testing?
Refers to testing the functions of an application that are visible to a user:

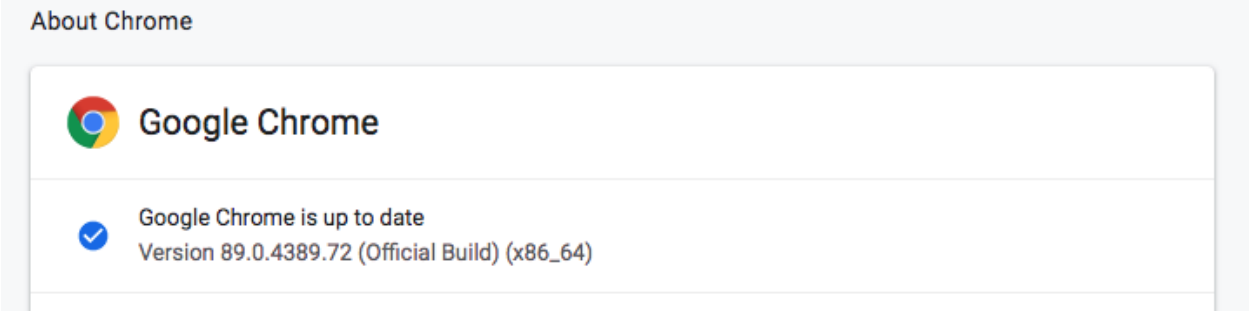
- Verifying that the application responds correctly to events such as clicking on the number and function buttons
- Confirming that appearance elements such as fonts and images conform to design specifications

A Tool for GUI Testing: Selenium WebDriver

- https://www.seleniumhq.org/projects/webdriver/
- Create robust, browser-based regression automation suites and tests
- Scale and distribute scripts across many environment
- Tutorial: https://www.guru99.com/selenium-tutorial.html

System Prerequisites:

1. **Chrome:**
   - Chrome at latest version (March 4th, 2021): **89.0.4389.72**.
   - Check your Chrome by select the three dots symbol on the top right, select **"Chrome"** then choose **"About Google Chrome"** in your Chrome browser. If your Chrome browser is old (not up to date), please upgrade.



2. **Chrome Selenium Web Driver for version 89.0.4389.23:**
   - https://chromedriver.storage.googleapis.com/index.html?path=89.0.4389.23/
3. **Selenium WebDriver Client for Java (Latest stable version 3.141.59):**
   - **https://www.seleniumhq.org/download/**

Selenium Client & WebDriver Language Bindings

In order to create scripts that interact with the Selenium Server (Selenium RC, Selenium Remote WebDriver) or create local Selenium WebDriver scripts, you need to make use of language-specific client drivers. These languages include both 1.x and 2.x style clients.

While language bindings for other languages exist, these are the core ones that are supported by the main project hosted on GitHub.

Language	Client Version	Release Date			
Java	3.141.59	2018-11-14	Download	Change log	Javadoc
C#	3.14.0	2018-08-02	Download	Change log	API docs
Ruby	3.14.0	2018-08-03	Download	Change log	API docs
Python	3.14.0	2018-08-02	Download	Change log	API docs
Javascript (Node)	4.0.0-alpha.1	2018-01-13	Download	Change log	API docs

Tutorial:

- https://wiki.saucelabs.com/display/DOCS/Getting+Started+with+Selenium+for+Automated+Website+Testing
- https://www.guru99.com/selenium-tutorial.html

Please download Lab06-TestingPlatforms-SampleCodes.zip and open the "Lab06-TestingPlatforms-SampleCodes/Selenium-Codes" folder.

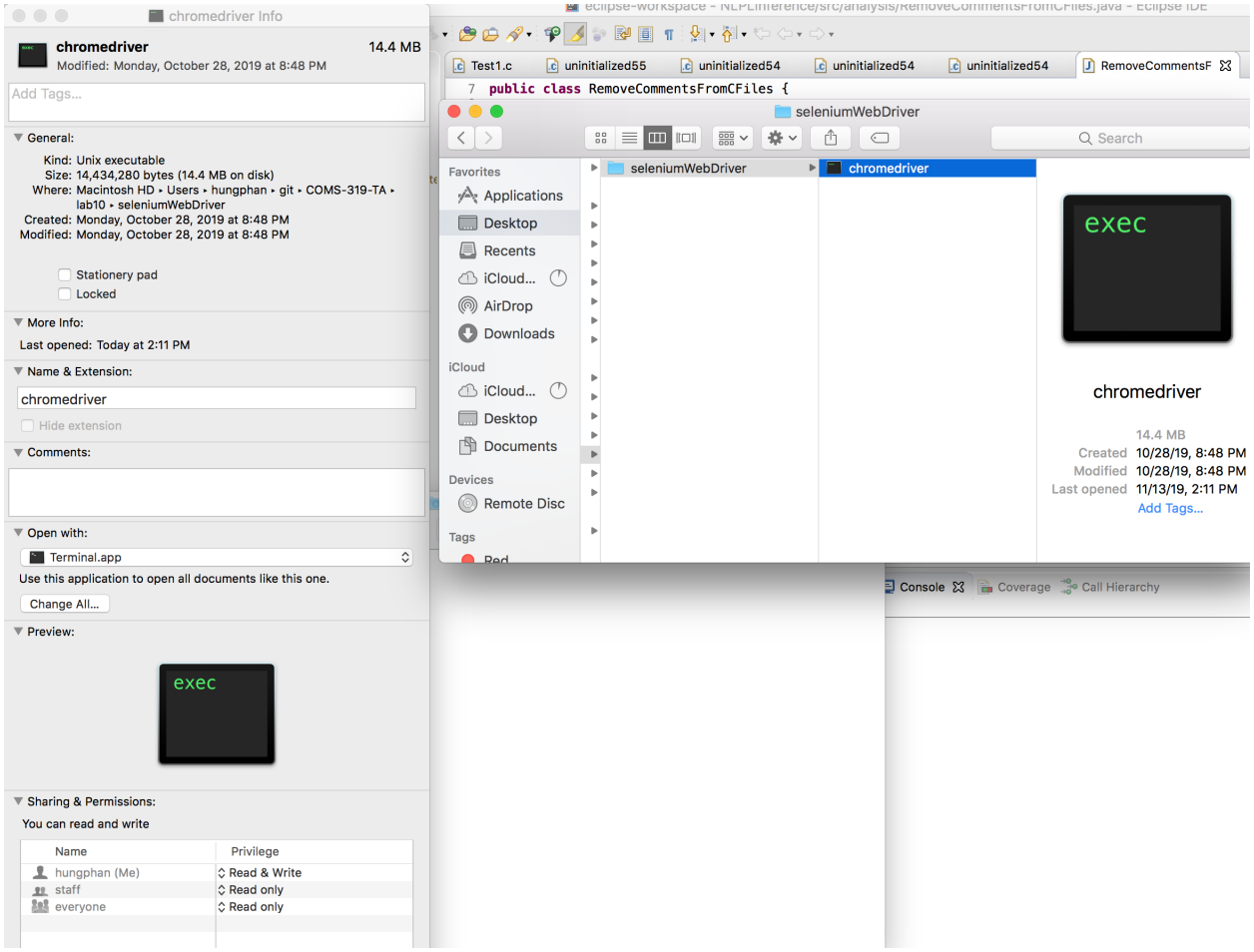
Task 1: Install Web Selenium Driver

If you already have Web Selenium and be able to run it on Java project, you can do from step 4.

Step 1: Download ChromeDriver

- https://chromedriver.chromium.org/downloads

Step 2: After downloading, you will have an execution file WebDriver. Put it in a specific folder in your computer. For example:



Save your {path to ChromeDriver file} to a specific file/document. On MacOS, a sample path for drive should be in this template:

/Users/hungphan/git/COMS-319-TA/Fall-2020/Lab6/driver/chromedriver

Note: On Windows, please the file name is **chromedriver.exe**

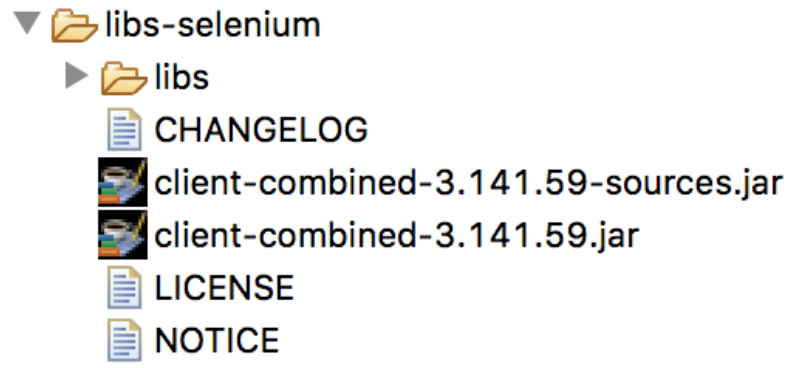
Step 3 (Optional): Append {Path of folder contained chromedriver} to your PATH environment variable.

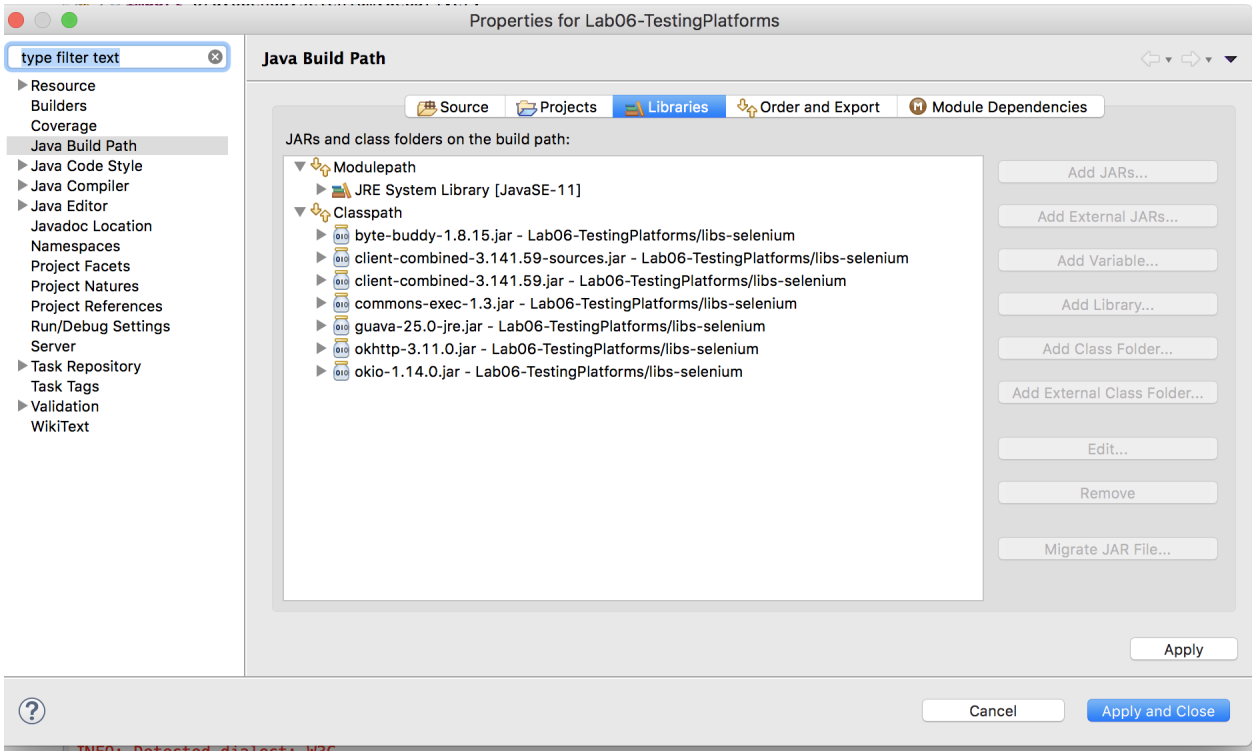
```
[hungs-mbp-4:Jest-Codes hungphan$ cd /Users/hungphan/git/COMS-319-TA/Fall-2020/La]
b6/driver/
[hungs-mbp-4:driver hungphan$ export PATH=$PATH:/Users/hungphan/git/COMS-319-TA/F]
all-2020/Lab6/driver/
hungs-mbp-4:driver hungphan$
```

- This will allow you to call APIs of Selenium without the need of adding by System.setProperty() in future.

Step 4: Create Java Project **Lab05Selenium** by Eclipse. (While creating the project, if you are asked about the creation of the module, select “Don’t Create”)

Step 5: Import and add all jar files in **libs-selenium** folder in the sample code (or you can download jar files from here: https://www.seleniumhq.org/download/). You are recommended to copy the selenium folder inside your project and add import all jar files (also in libs folder).





Step 6: Import **CheckSelenium.java** inside ‘Selenium-Codes’ to your Java project.

Step 7: Edit **CheckSelenium.java** by update your {Path to ChromeDriver} in line 9.

```java
package lab10;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class CheckSelenium {
    public static void main(String[] args) {
//      Set system property (change your path in second arguments
        System.setProperty("webdriver.chrome.driver",
                "/Users/hungphan/git/COMS-319-TA/lab10/seleniumWebDriver/chromedriver");
        WebDriver driver = new ChromeDriver();
        String baseUrl = "https://www.google.com";
        String expectedTitle = "Google";
        String actualTitle = "";
        // launch Chrome and direct it to the Base URL
        driver.get(baseUrl);
        // get the actual value of the title
        actualTitle = driver.getTitle();
        if (actualTitle.contentEquals(expectedTitle)) {
            System.out.println("Test Passed!");
        } else {
            System.out.println("Test Failed!");
        }
        // close Fire fox
        driver.close();

    }
}
```

Step 8: Run **CheckSelenium.java.** You should see this output.

```
<terminated> CheckSelenium [Java Application] /Library/Java/JavaVirtualMachines/jdk-11.0.3.jdk/Contents/Home/bin/java (Nov 14, 2019, 10:11:44 AM)
Starting ChromeDriver 78.0.3904.70 (edb9c9f3de0247fd912a77b7f6cae7447f6d3ad5-refs/branch-heads/3904@{#800}) on port 14524
Only local connections are allowed.
Please protect ports used by ChromeDriver and related test frameworks to prevent access by malicious code.
Nov 14, 2019 10:11:45 AM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
Test Passed!
```

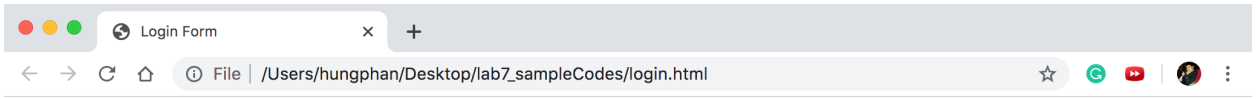
Task 2: Login GUI Testing.

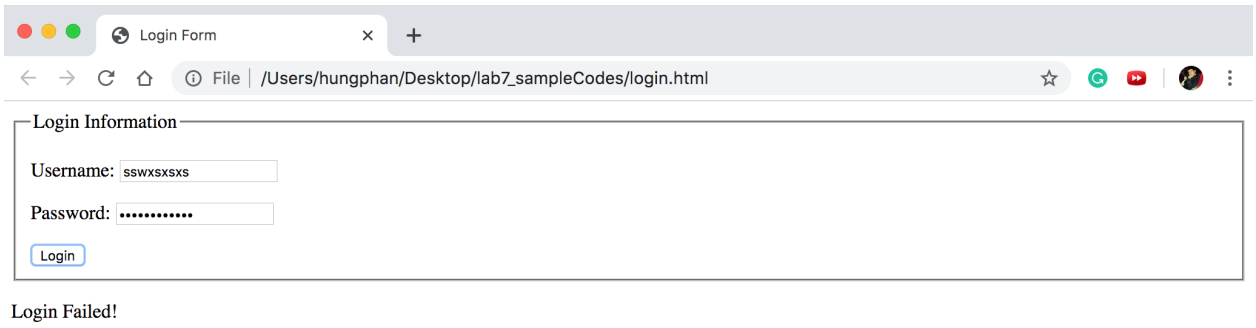
In this task, you will make 2 unit tests for automatically test login page passed or failed. We simulates 2 cases of login:

1. If users login by username **“coms319”** and password **“lab10”**, the login will success. **This is the only username and password accepted for login.**
2. If users login by other username or password, such as by username “test” and password “lab12”, the login will fail.

You are provided the login.html and login.js and you don’t need to modify these files.

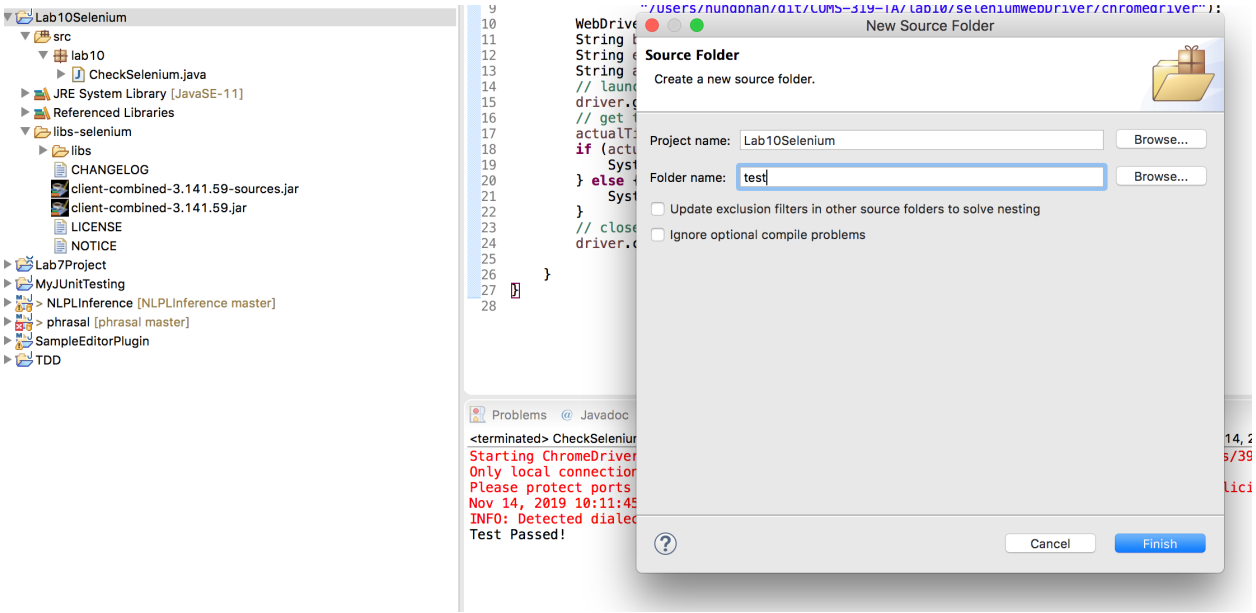
Step 1: View the file login.html by Chrome. Login by 2 usernames and passwords above. You should see 2 outputs as follows:





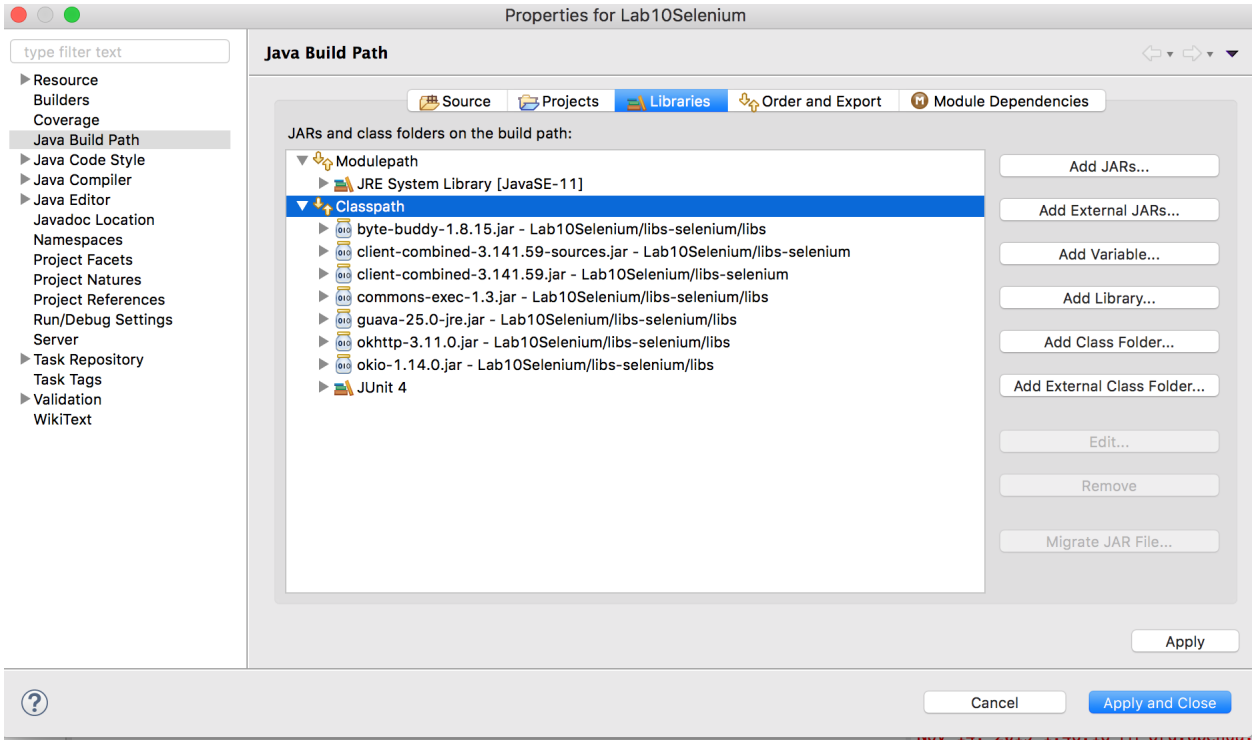
Step 2: Read login.html and login.js. Look at the "id" of each HTML elements and understand their roles in HTML and Javascript.

Step 3: Now create unit test for Login Pass. First, create test src folder:



Step 4: Import Junit4 to your project

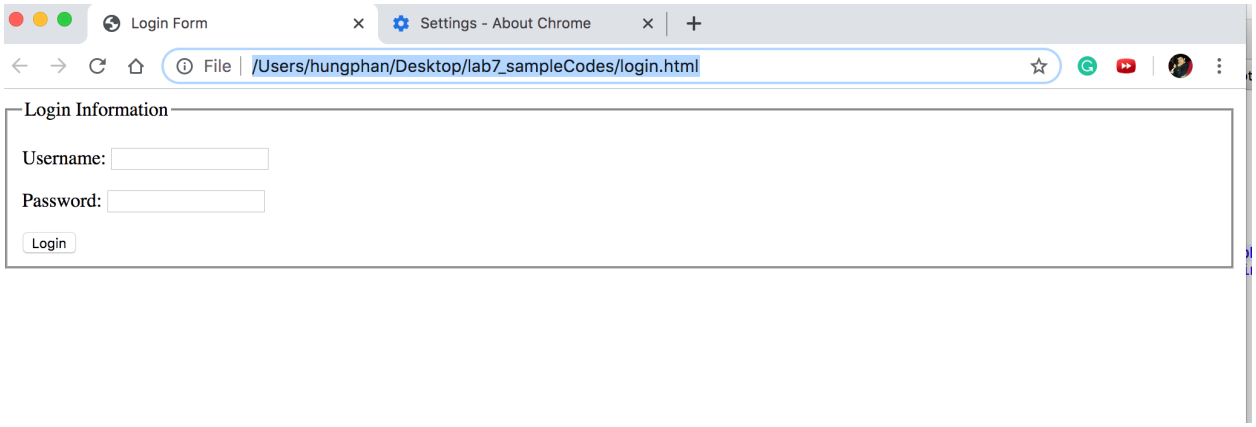
(Java Build Path ➡ Libraries ➡ Add Library ➡JUnit ➡JUnit4)



Step 5: Create package **lab06** inside Test folder and copy **LoginTest** in sample code folder to this package.

Step 6: Change **pathLoginPage**'s value to **web url of your login.html** you opened in step 1 and change **pathChromeDriver**'s value to the **{path to ChromeDriver}** you have.

- You can see the web url of login.html by checking your Chrome bar.



Edit these paths:

```
17
18 //  Change your selenium driver path here
19     static String pathChromeDriver="/Users/hungphan/git/COMS-319-TA/Fall-2020/Lab6/driver/chromedriver";
20     static String pathLoginPage="file:///Users/hungphan/git/COMS-319-TA/Fall-2020/Lab6/solution/Selenium-Codes/login.html"
21
```

Step 6: Now in LoginTest.java you see 2 test cases for login success and login failed. Before implementing these test cases, you need to initiate the WebDriver before executing test cases. After running the test cases, you need to close the web browser.You do that by implementing @BeforeClass function and @AfterClass function. Uncomment the content of function **openBrowser** and **closeBrowser.**

```
27      @BeforeClass
28      public static void openBrowser()
29      {
30          System.setProperty("webdriver.chrome.driver", pathChromeDriver);
31          driver= new ChromeDriver() ;
32          driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
33      }
34
35      @AfterClass
36      public static void closeBrowser() {
37          driver.quit();
38      }
```

Step 7: Implement the test case for login success as follows (uncomment the commented code inside this test case):

```
40      @Test
41      public void loginSuccessTest() throws InterruptedException {
42          driver.get(pathLoginPage);
43          driver.manage().window().maximize();
44          driver.findElement(By.xpath("//input[@id='"+txtUsername+"']")).sendKeys("coms319");
45          driver.findElement(By.xpath("//input[@id='"+txtPassword+"']")).sendKeys("lab10");
46
47          Thread.sleep(1000);
48          driver.findElement(By.xpath("//input[@id='"+btnLogin+"']")).click();
49
50          String strMessage=driver.findElement(By.xpath("//label[@id='"+txtMessageLogin+"']")).getText();
51          assertEquals("Failed test case", strMessage, "Login Success!");
52      }
```

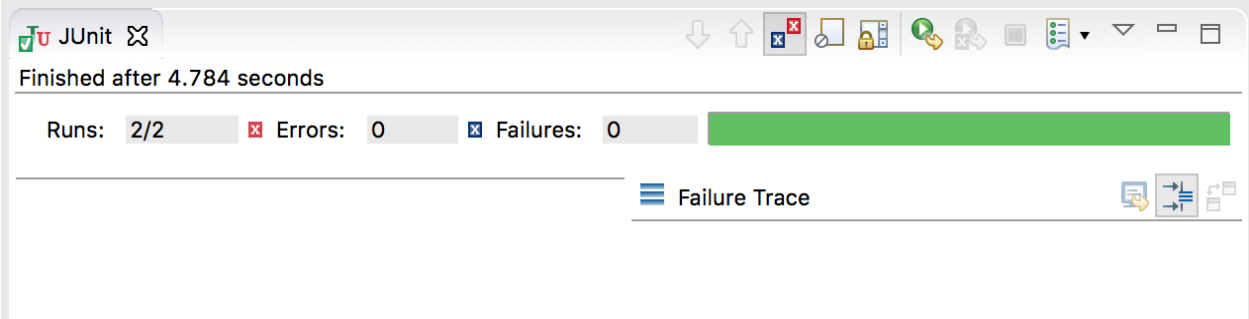
- In the code above, you use Selenium APIs to find HTML elements by their ID in line 44 and 45. You use XPath strategy to find and do actions to send input to 2 text boxes.

Step 8: Implement the test case for login failed as follows (uncomment the commented code inside this test case):

```
54      @Test
55      public void loginFailedTest() throws InterruptedException {
56          driver.get(pathLoginPage);
57          driver.manage().window().maximize();
58          driver.findElement(By.id(txtUsername)).sendKeys("test");
59          driver.findElement(By.id(txtPassword)).sendKeys("lab10");
60
61          Thread.sleep(1000);
62          driver.findElement(By.id(btnLogin)).click();
63          String strMessage=driver.findElement(By.xpath("//label[@id='"+txtMessageLogin+"']")).getText();
64          assertEquals("Failed test case", strMessage, "Login Failed!");
65      }
```

- Along with XPath, you can also use By.id to find and do action to elements.
- After simulating input of users, you perform click button and check the message label to see if it is login success or failed.

Step 9: Run the test cases. You will see the web browser automatically open to login.html, perform login and provide the output. After each test case, the Chrome browser is close. You should see both test cases passed.



You have completed unit test functions for Web GUI using Selenium WebDriver.