COMMITTED TO EXCELLENCE IN EDUCATION

निर्मलस्नेह उत्तम सेवाधर्म

SHRI G.P.M. DEGREE COLLEGE OF SCIENCE & COMMERCE

SHRI G.P.M. DEGREE COLLEGE OF SCIENCE & COMMERCE.

(COMMITED TO EXCELLENCE IN EDUCATION)

CERTIFICATE

This is to certify that Mr/Ms._____

a student of TY**.BSC-CS** Roll no:_____has completed the required number of practicals in the subject of_____

as prescribed by the UNIVERSITY OF MUMBAI under my supervision during the academic year 2024-2025.

…….. ……

Prof. Incharge Principal

…… …..…..……

External Examiner Course Co-ordinator

…….. …….

Date College Stamp

Professor Name: Swaraj Humbare	Class : TY.BSC-CS Semester : Sem V (2024-2025)
Course code : USCS503	Subject: Software Testing And Quality Assurance

Index

Sr. No.	Date		Page No.	Sign
1		**Practical 1**: Install Selenium IDE and create a test suite containing a minimum of 4 test cases for different web page formats (e.g., HTML, XML, JSON, etc.)	1 – 14	
2		**Practical 2:** Conduct a test suite for two different websites using Selenium IDE, Perform various actions like clicking links, filling forms, and verifying content	15 – 25	
3		**Practical 3 :** Install Selenium Server (Selenium RC) and demonstrate its usage by executing a script in Java to automate browser actions.	26 – 36	
4		**Practical 4:** Write a program using Selenium WebDriver to update 10 student records in an Excel file. Perform data manipulation and verification.	37 – 44	
5		**Practical 5:** Write a program using Selenium WebDriver to select the number of students who have scored more than 60 in any one subject (or all subjects). Perform data extraction and analysis.	45 – 52	
6		**Practical 6:** Write a program using Selenium WebDriver to provide the total number of objects present or available on a web page. Perform object identification and counting.	53 – 59	
7		**Practical 7**: Write a program using Selenium WebDriver to get the number of items in a list or combo box on a web page. Perform element identification and counting	60 – 67	
8		**Practical 8**: Write a program using Selenium WebDriver to count the number of checkboxes on a web page, including checked and unchecked counts. Perform checkbox identification and counting.	68 - 75	

****

Theory 1: Selenium Installation IDE

Selenium IDE (Integrated Development Environment) is a tool used for automating web browser interactions. It allows users to create test scripts by recording their interactions with a web application. Here's a brief theory behind installing and using Selenium IDE:

1. Overview of Selenium IDE

- Selenium IDE is an open-source automation testing tool, originally developed for Mozilla Firefox. It provides a playback and recording tool for authoring tests without learning a scripting language.
- It allows users to record their actions on a web application and generate test scripts in various programming languages.
- It is a browser extension available for both Chrome and Firefox.

2. Why Use Selenium IDE?

- **Ease of Use**: It enables users to record browser actions and generate test scripts quickly without requiring extensive programming knowledge.
- **Playback Feature**: The recorded tests can be replayed to verify the consistency of web application behaviors across different versions.
- **Script Editing**: Users can edit the recorded test scripts manually for custom automation tasks.

3. Installation Steps for Selenium IDE

- **Step 1: Download and Install the Browser Extension**
  - Go to the Chrome Web Store or Mozilla Add-ons website.
  - Search for "Selenium IDE" and click "Add to Chrome" or "Add to Firefox" to install the extension.
- **Step 2: Launch the Selenium IDE**
  - After installation, a new Selenium IDE icon will appear in the browser toolbar. Click it to launch the IDE.
- **Step 3: Configure Basic Settings**
  - Once launched, the IDE provides options to start a new project, open an existing one, or start recording a new test.

4. Features of Selenium IDE

- **Recording and Playback**: Automatically records user interactions with the browser and can replay the recorded actions.
- **Test Case Creation**: Allows users to create multiple test cases, organize them into test suites, and execute them.

- **Command Editing**: Users can edit recorded commands, insert breakpoints, and customize test scripts.
- **Integration**: Supports exporting test cases to Selenium WebDriver code, enabling integration with different programming environments (e.g., Java, Python).

5. Using Selenium IDE

- **Recording a Test**: To record a test, start the recording session and perform the desired browser actions. Selenium IDE will capture each step.
- **Playing Back Tests**: Run the recorded test to see if it successfully replicates the desired user interactions.
- **Debugging**: Use breakpoints and the log viewer to identify and fix any issues in the test scripts.

6. Advantages of Selenium IDE

- Quick setup without complex configurations.
- No programming skills required for basic recording and playback.
- Useful for learning the basics of test automation.
- Can be used as a starting point for more advanced Selenium WebDriver projects.

7. Limitations

- Not as flexible or powerful as Selenium WebDriver for complex test scenarios.
- Can only automate actions within supported web browsers.
- Lacks some advanced features for handling non-web-based applications or sophisticated workflows.

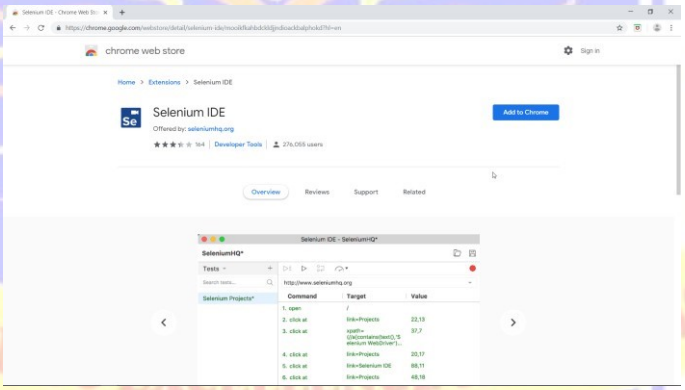
Practical 1: Install Selenium IDE and create a test suite containing a minimum of 4 test cases for different web page formats (e.g., HTML, XML, JSON, etc.)

REQUIREMENTS:

1) Selenium IDE extension on a browser.
2) NetBeans IDE for creating test case files.

STEPS:

1) Just **google "Selenium IDE chrome"** and click on the first link. **Add** the **"Selenium IDE" extension** to your browser.



2) For testing, we need a **test case suite**, so we create a **WebApplication**

in **NetBeans IDE**.**File > New Project > Java Web > Web Application > Add GlassFish Server > Finish.**

3) In *index.html*, we **write the following code** for "ARITHMETIC OPERATIONS" test case suite:

---index.html---

```
<html>
  <head>
    <title>prac1</title>
    <script
      language="
      javascript">
      function
      addition()
        {
          var
          num1=parseInt(document.arithme
          tic.n1.value);var
          num2=parseInt(document.arithme
          tic.n2.value);var
          result=num1+num2;
```

```
    document.arithmetic.res.value=res
    ult;

  }

function subtraction()

  {

    var
    num1=parseInt(document.arithme
    tic.n1.value);var
    num2=parseInt(document.arithme
    tic.n2.value);var result=num1-
    num2;
    document.arithmetic.res.value=res
    ult;

  }

function multiplication()
  {

    var
    num1=parseInt(document.arithme
    tic.n1.value);var
    num2=parseInt(document.arithme
    tic.n2.value);var
    result=num1*num2;
    document.arithmetic.res.value=res
    ult;

  }

function division()

  {

    var
    num1=parseInt(document.arithme
    tic.n1.value);var
    num2=parseInt(document.arithme
    tic.n2.value);var
    result=num1/num2;
    document.arithmetic.res.value=res
    ult;

  }
</script>
```

```
</head>
<body>
  <h1 align="center"> Arithmetic Operations </h1>
  <form name="arithmetic">
    <table border="1" align="center">
      <tr>
        <td>Number 1: </td>
        <td><input type="text" name="n1" size="20"></td>
      </tr>
      <tr>
        <td>Number 2: </td>
        <td><input type="text" name="n2" size="20"></td>
      </tr>
      <tr>
        <td colspan=2>
          <input type="button"
name="add" value="Add"
onclick="javascript:addition();"> 
          <input type="button" name="sub"
value="Subtract"
onclick="javascript:subtraction();"> 
          <input type="button" name="mul"
value="Multiply"
onclick="javascript:multiplication();"> 
          <input type="button"
name="div" value="Divide"
onclick="javascript:division();"> 
        </td>
      </tr>
      <tr>
          <td colspan="2">Result is: <input type="text" name="res" size="20"
value=""></td>
      </tr>
```

```
        </table>
      </form>
    </body>
  </html>
```

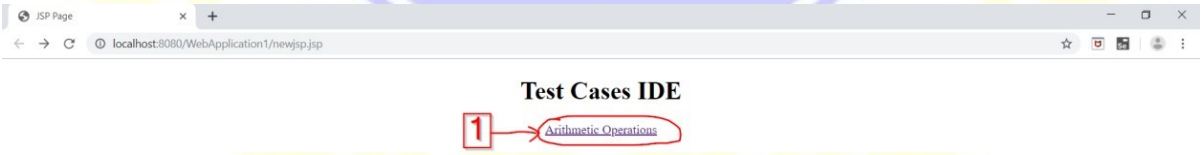
4) Create a **JSP file** *"newjsp.jsp"* with a link to the created HTML file.

---newjsp.jsp---

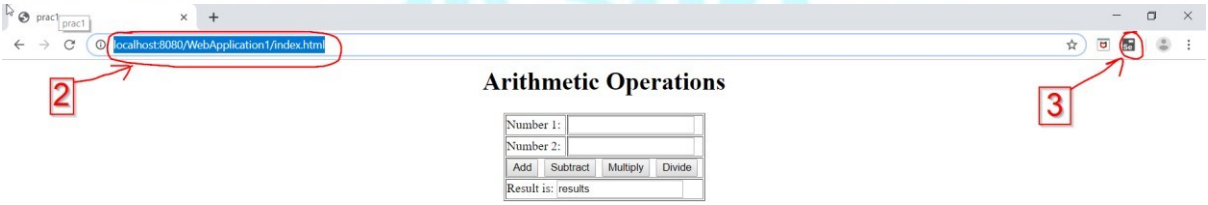
```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>JSP Page</title>
  </head>
  <body align="center">
    <h1>Test Cases IDE</h1>
    <a href="index.html">Arithmetic Operations</a>
  </body>
</html>
```

5) **Run** this JSP file from NetBeans. It will then open up in a browser.

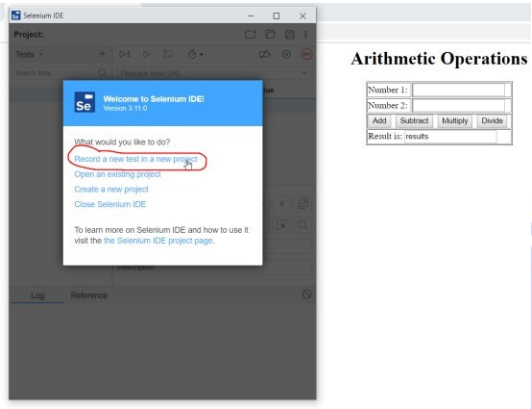
- Then from the browser, **open the link**:



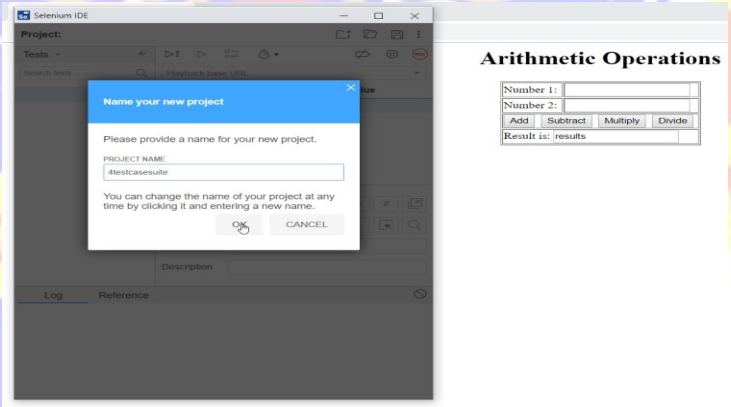
- Now you'll be redirected to ARITHMETIC OPERATIONS page.
  Copy the URL of the page and **open "Selenium IDE" extension**:



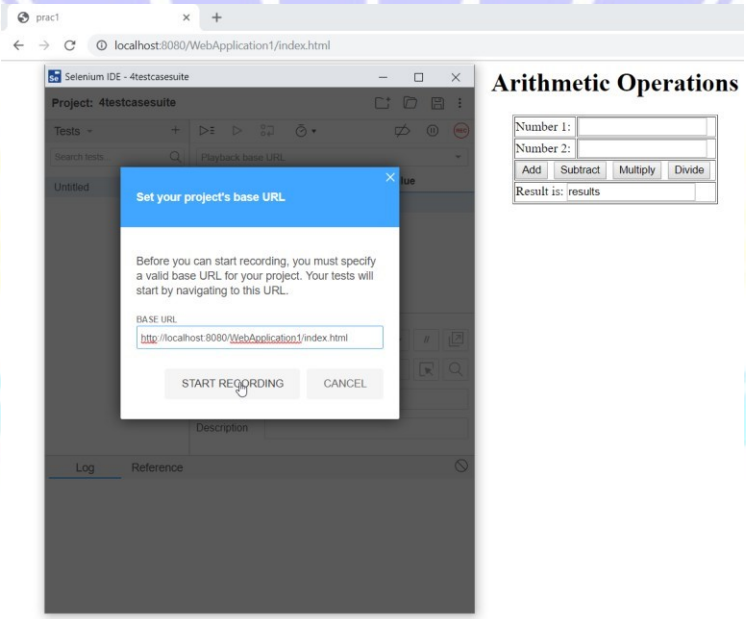
- Now, **Record a new test project in a new project:**



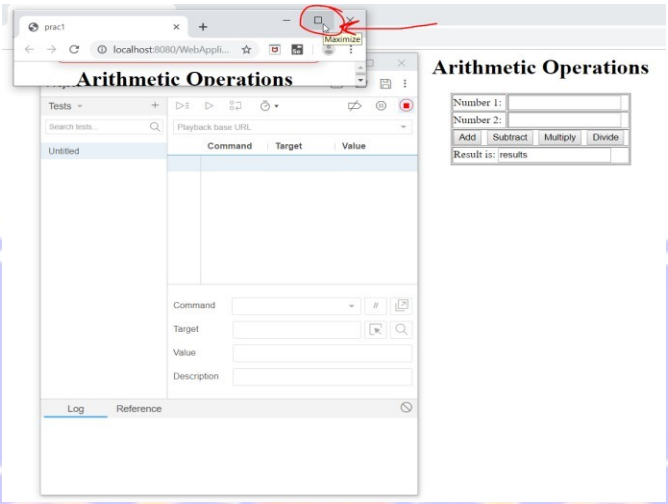
- **Name your project**:



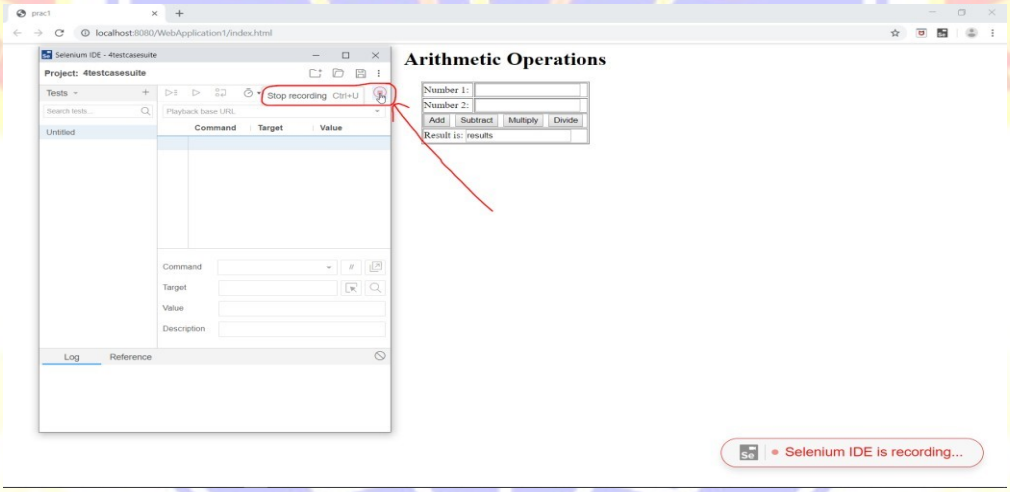
- Paste that URL here and click Start Recording:



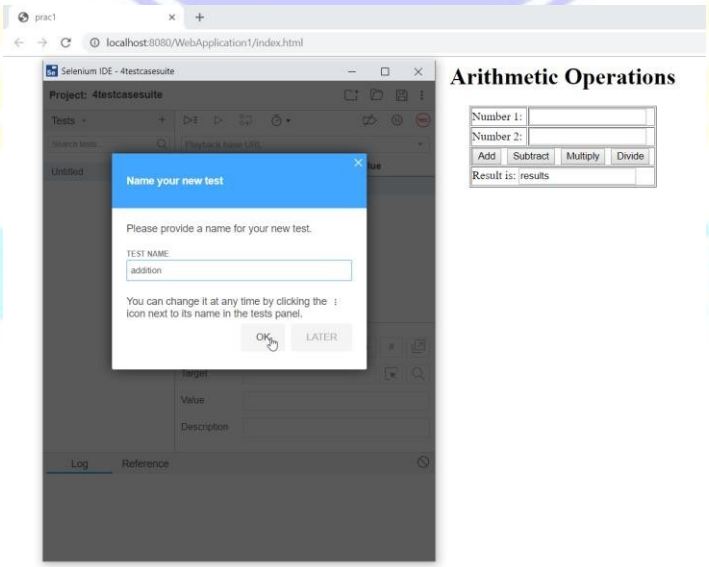
- A **new tab** will be opened, **maximize** that tab(this tab is your working tab for recording):



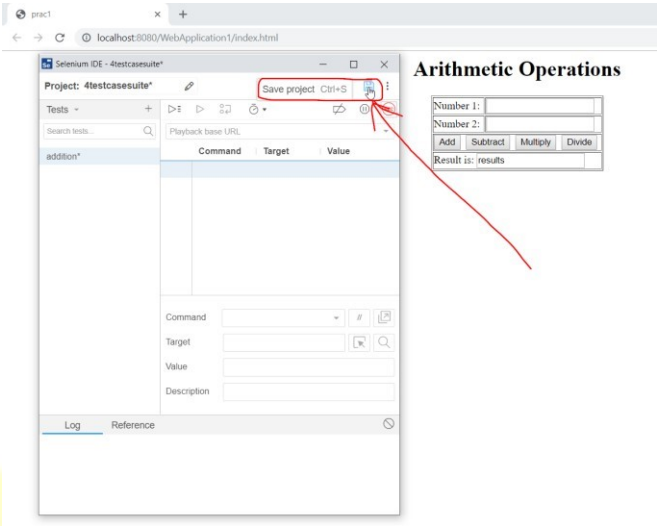
- Click on **Stop Recording**:



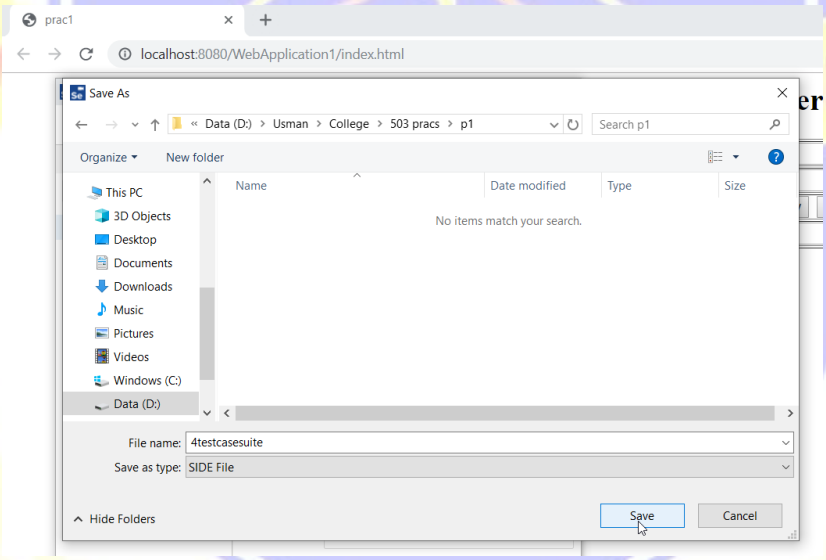
- Just as you stop your recording, you'll be prompted to **Name your new test,** we'll makeour new test as "**addition**":



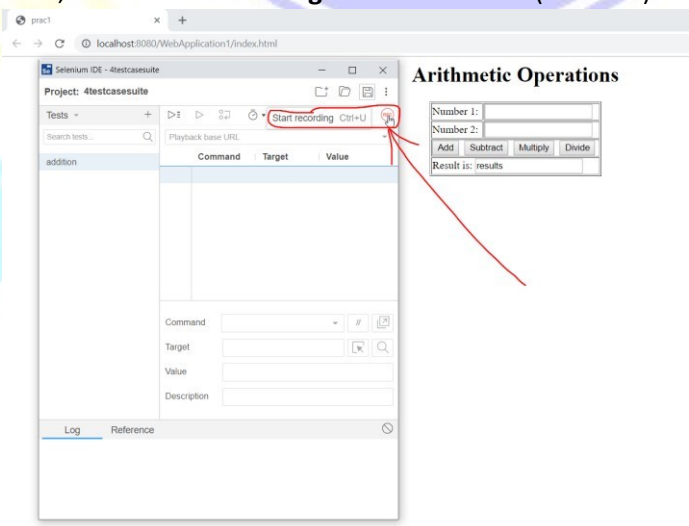
- **Save the project** after each test case:



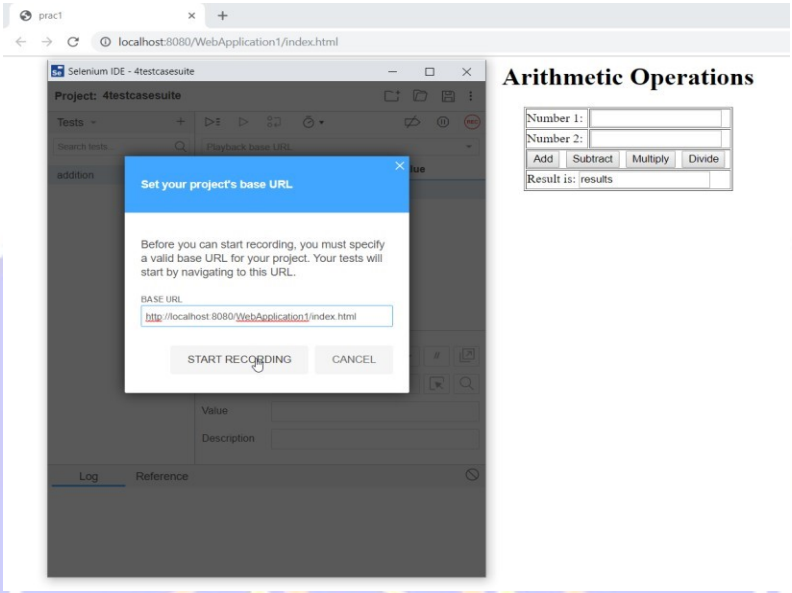
- A file with "**.SIDE**" extension will be saved(SIDE abbrev to **S**elenium **IDE**):



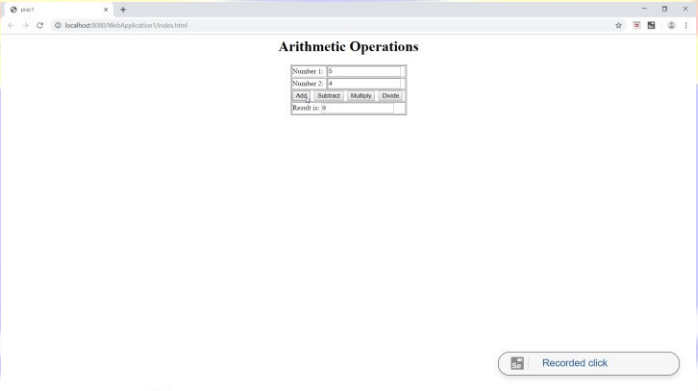
- Now, we'll **Start Recording** our first test case(addition):



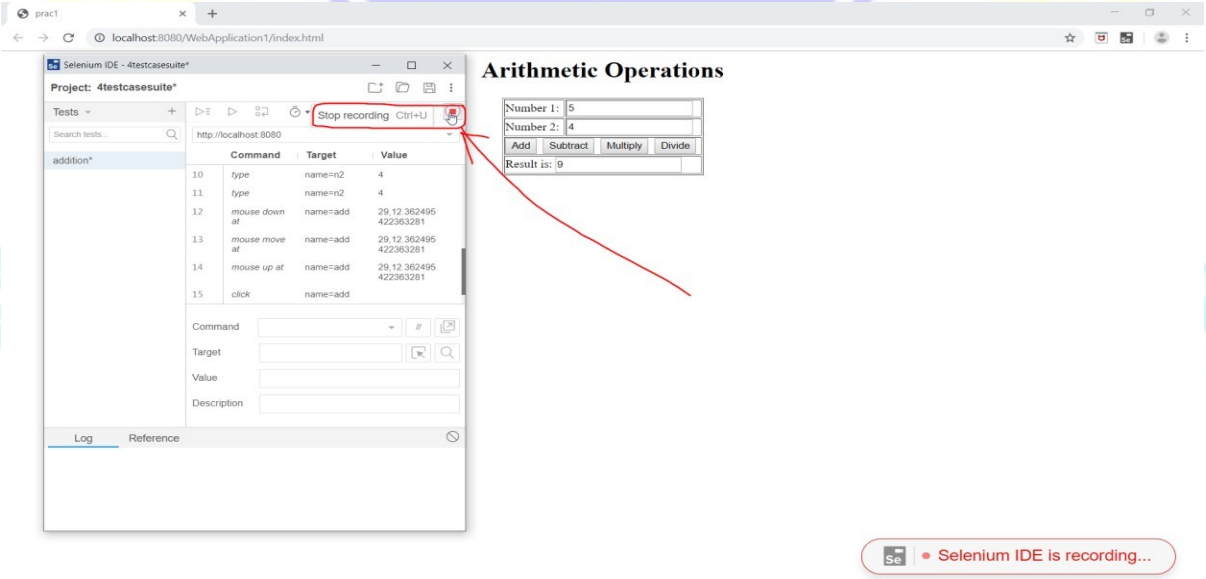
- We'll be prompted to **paste our URL again**, then click **Start Recording:**



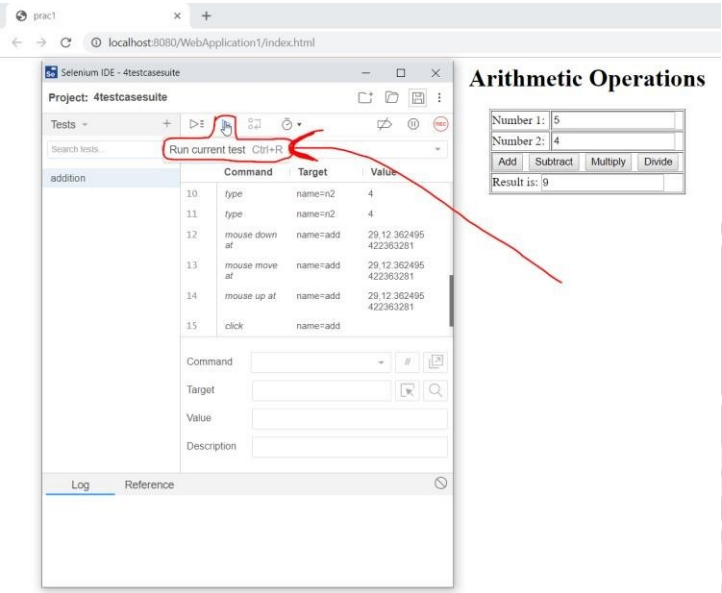
- **NOTE THAT** each clicks are recorded:



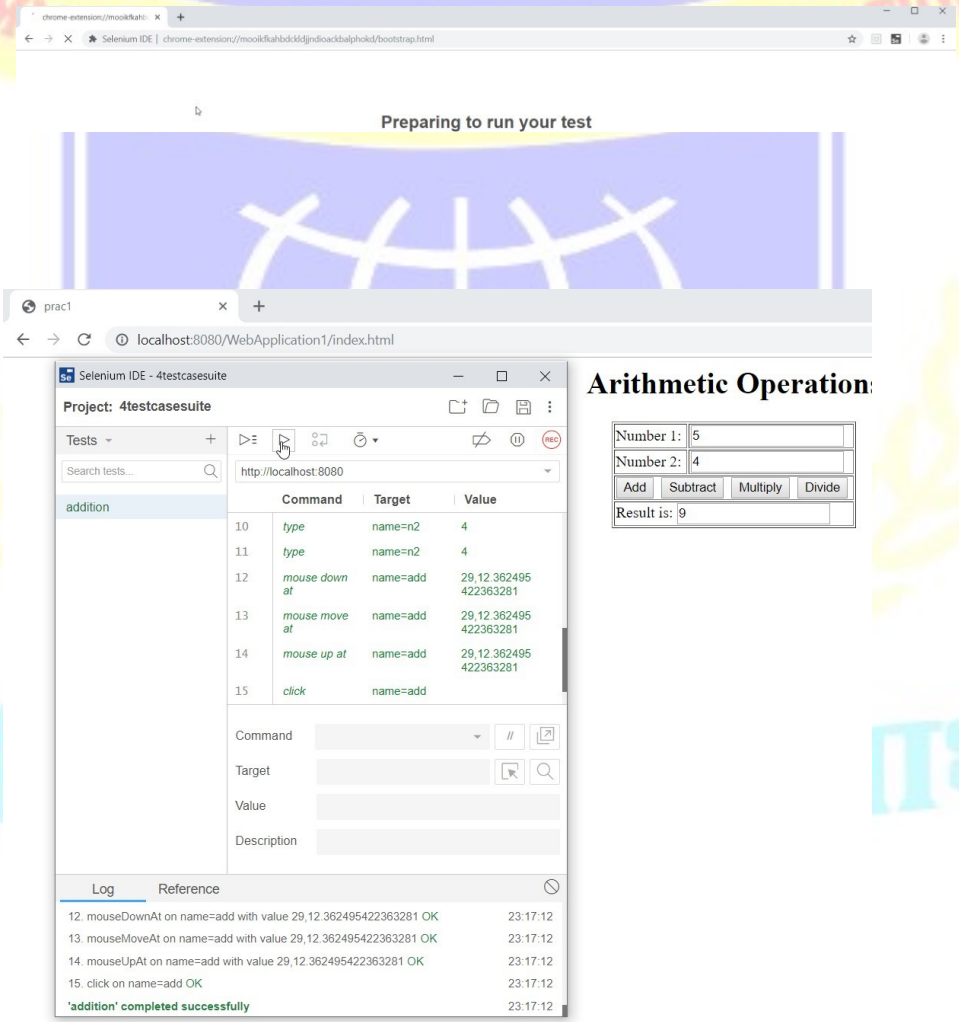
- Now we'll **Stop Recording from the Selenium IDE:**



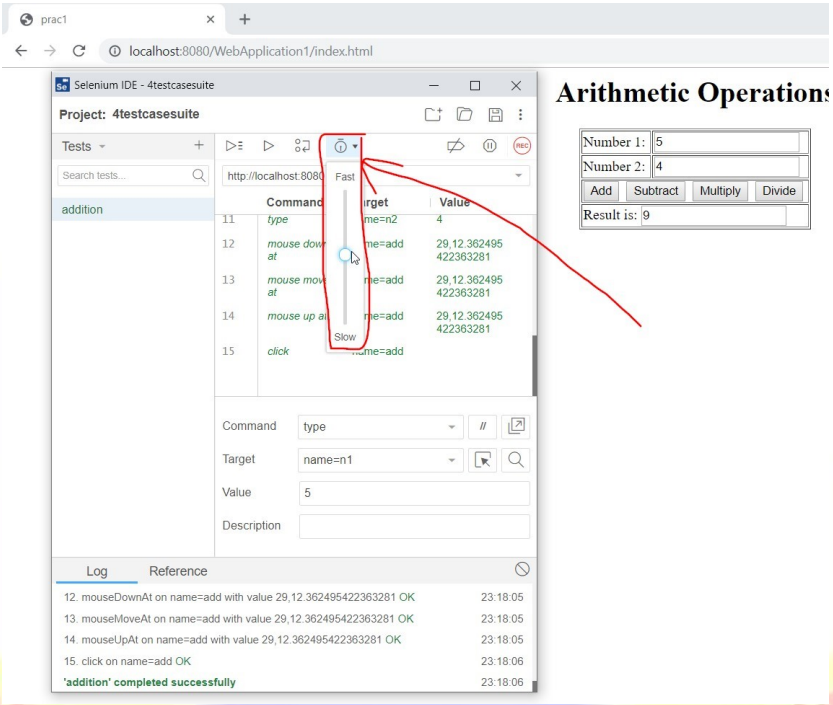
- We'll then **Run current test**(addition):



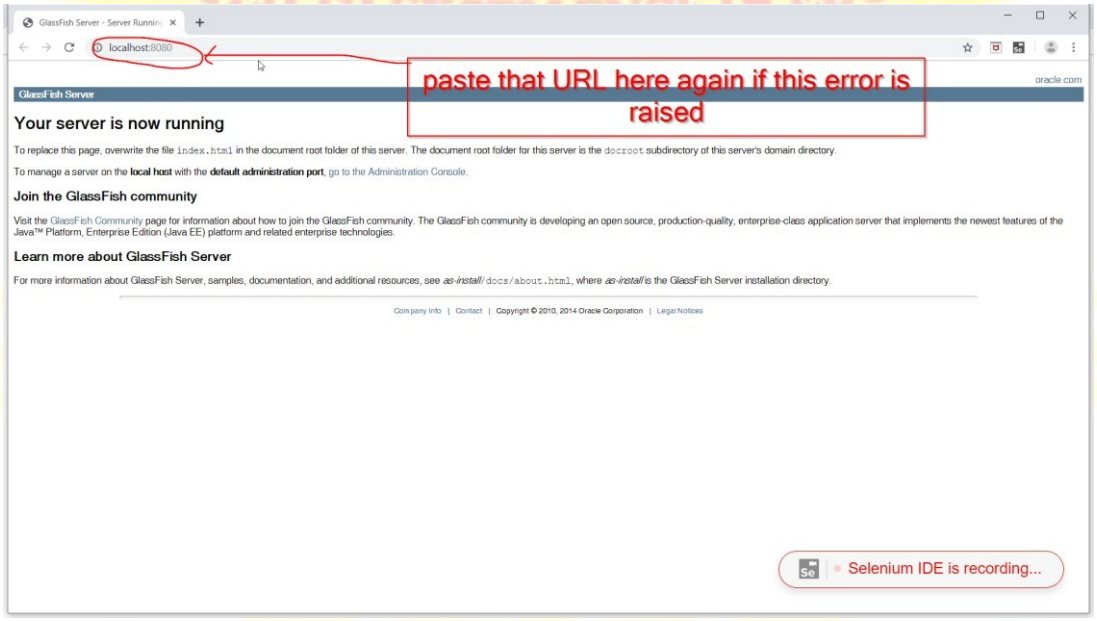
- The **test will run automatically** and the **completion will be shown step-by-step** in the **"Log" area** in the Selenium IDE:



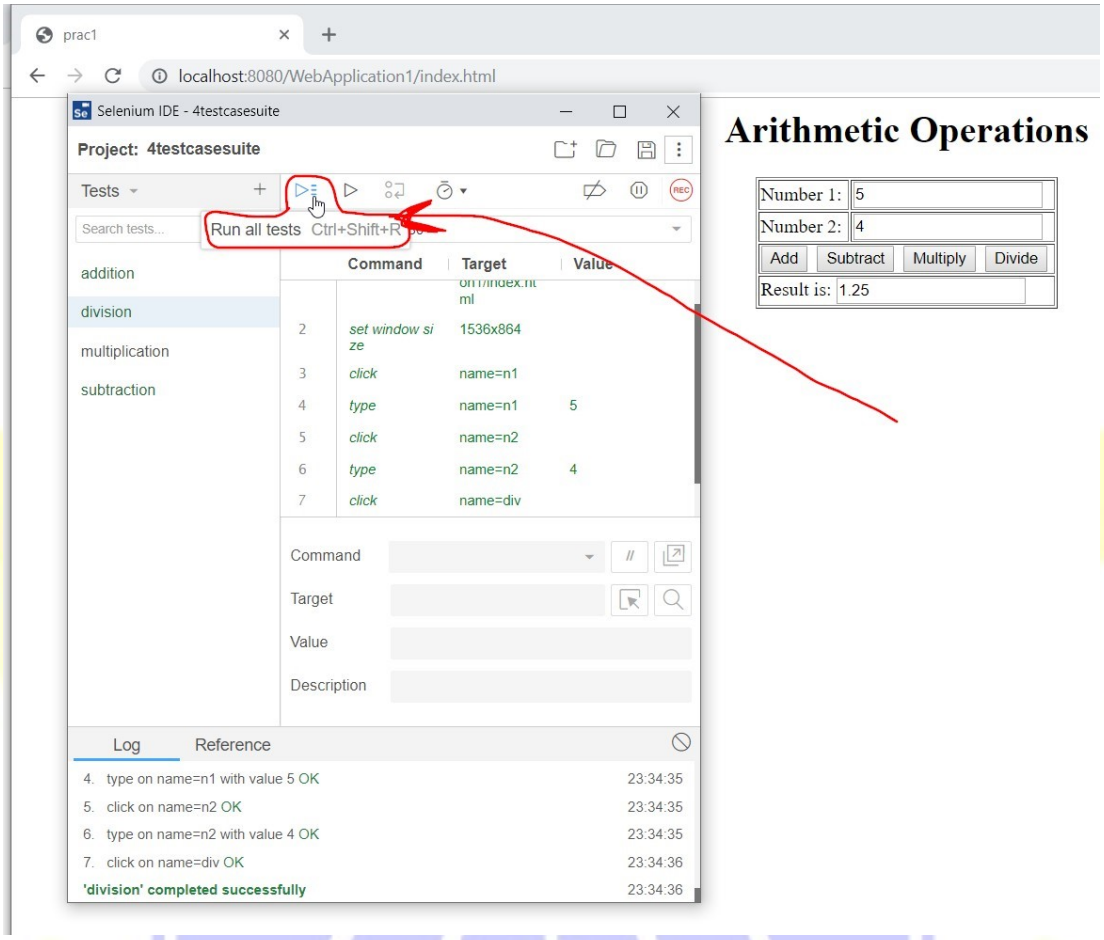
- **NOTE1: You can set the "Test execution speed" too:**



- **NOTE2: Paste here the URL again if the Glassfish Error occurs** between any test case**:**



- Now **create 3 more test cases**(subtraction, multiplication, division). Then "**Run all tests**":



6) Finish!

Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) What is Selenium IDE, and what is it used for?

2) How do you install Selenium IDE in a browser like Chrome or Firefox?

3) What types of web page formats can Selenium IDE test, and why is this important?

4) What is the role of a test suite in Selenium IDE, and how do you create one?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 2: Selenium IDE

A Selenium IDE test suite is a collection of test cases grouped together for sequential execution. It allows users to run multiple test cases in a specified order to test different parts of a web application or to execute end-to-end testing scenarios. Here's an overview of the details related to Selenium IDE test suites:

1. Understanding Test Suites

- A test suite in Selenium IDE consists of a set of test cases that are executed one after the other.
- Each test case contains a series of recorded commands or user actions to automate browser interaction.
- A test suite helps automate the testing of multiple workflows or functionalities without manually running individual test cases.

2. Creating a Test Suite in Selenium IDE

- Step 1: Create Individual Test Cases
  - Before creating a test suite, ensure that you have already recorded or created the individual test cases.
  - Test cases can include different actions, such as clicking buttons, filling out forms, navigating pages, etc.
- Step 2: Organize Test Cases into a Suite
  - Click on the "Add Suite" option in the Selenium IDE interface.
  - Name the test suite and add the existing test cases to the suite.
  - Test cases can be added by dragging and dropping them into the test suite, or through the "Add Test" option.
- Step 3: Set Execution Order
  - Arrange the order of test cases by dragging them up or down in the test suite. This determines the sequence in which the test cases will be executed.

3. Running a Test Suite

- To run a test suite, select it and click the "Run All" button.
- The test suite will execute each test case sequentially, following the specified order.
- Selenium IDE provides feedback on the execution of each test case, showing whether it passed or failed, along with log details for debugging.
- Users can also choose to pause, stop, or resume the execution of the test suite during runtime.

4. Test Suite Features in Selenium IDE

- Batch Execution: Test suites enable batch execution of test cases, automating several scenarios in one go.
- Test Case Reuse: The same test case can be added to multiple test suites, allowing for reuse in different workflows.
- Parallel Execution: If Selenium IDE supports a test runner that can execute test cases in parallel, test suites can be configured to run simultaneously (this may require additional setup).
- Result Reporting: Selenium IDE provides a summary report after running the test suite, showing which test cases passed, failed, or were skipped.

COMMITTED TO EXCELLENCE IN EDUCATION
निर्मलस्नेह उत्तम सेवाधर्म

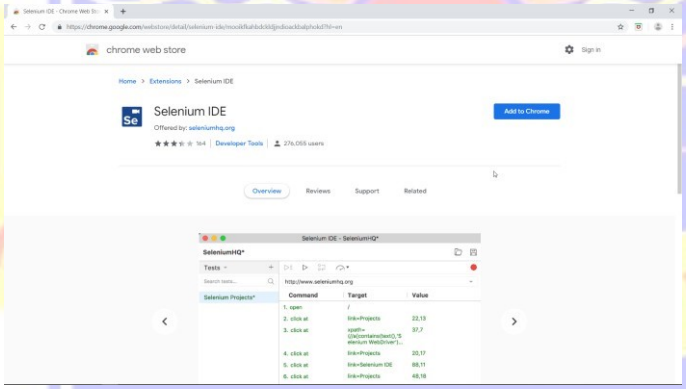
Practical 2: Conduct a test suite for two different websites using Selenium IDE, Perform various actions like clicking links, filling forms, and verifying content.

REQUIREMENTS:

1) Selenium IDE extension on a browser.
2) A stable Internet connection.

STEPS:

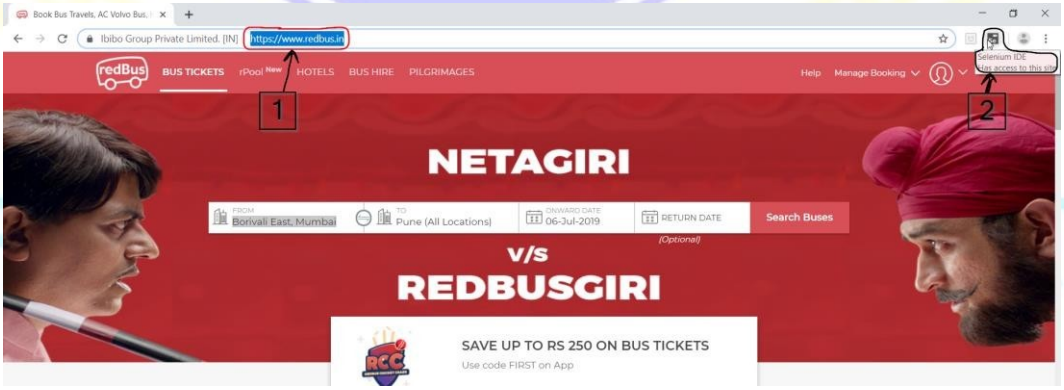
1) Just **google "Selenium IDE chrome"** and click on the first link. **Add** the **"Selenium IDE" extension** to your browser.



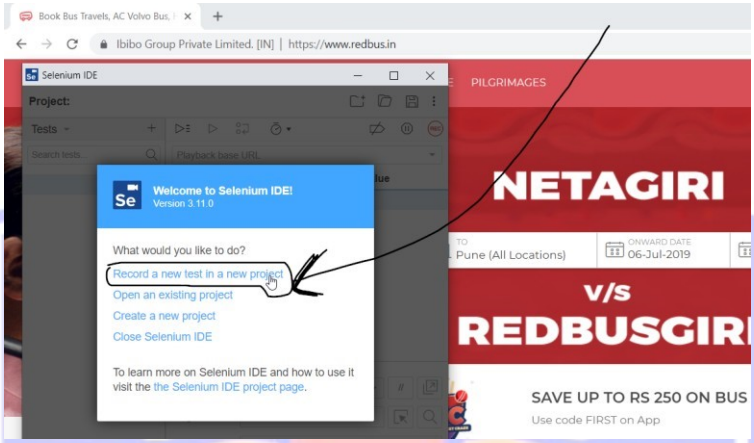
2) For testing, we'll choose "**redbus.in**" as our first website. We'll be checking some **SeleniumIDE Commands** like **assertText, verifyTitle, storeText, echo**. Detailed info for several commands are provided at https://ui.vision/docs/selenium-ide .

- Now **google "redbus.in".**

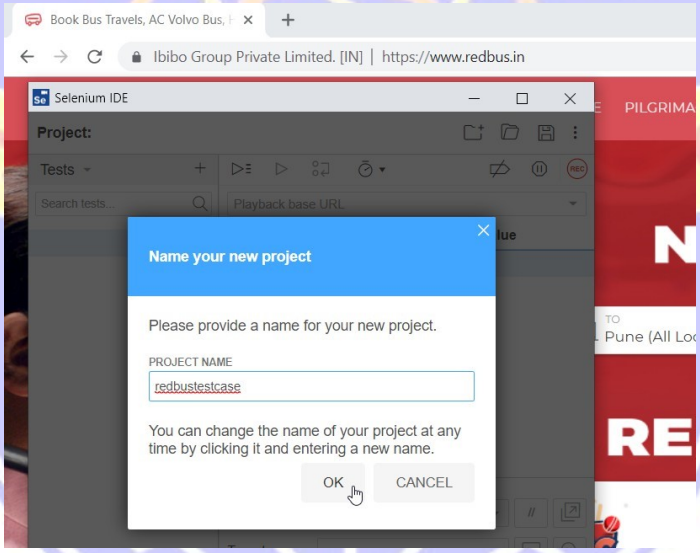
- **Copy the URL** of the page and **open "Selenium IDE" extension**:



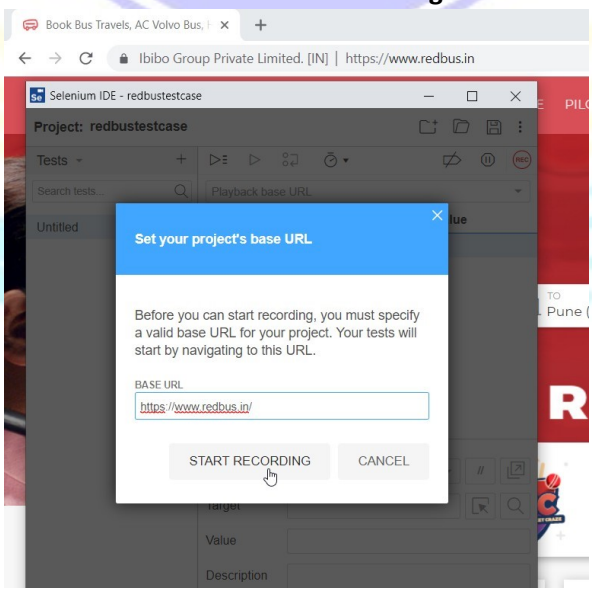
- Now, **Record a new test project in a new project:**



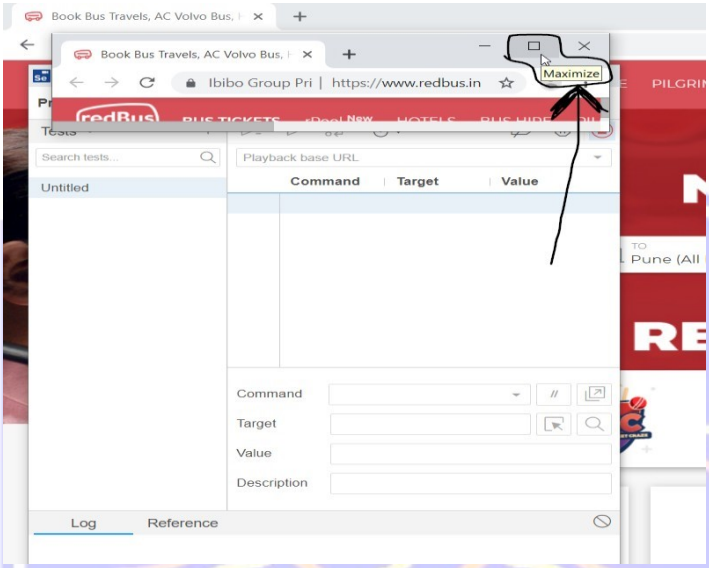
- **Name your project**:



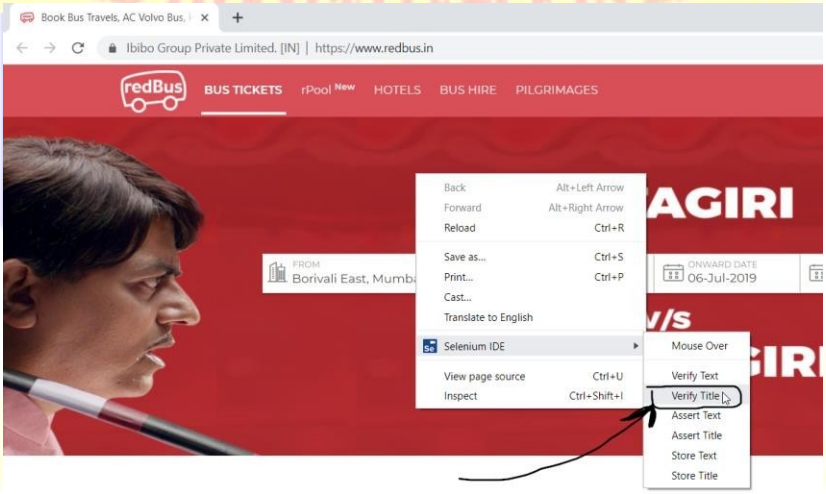
- **Paste that URL** here and click **Start Recording:**



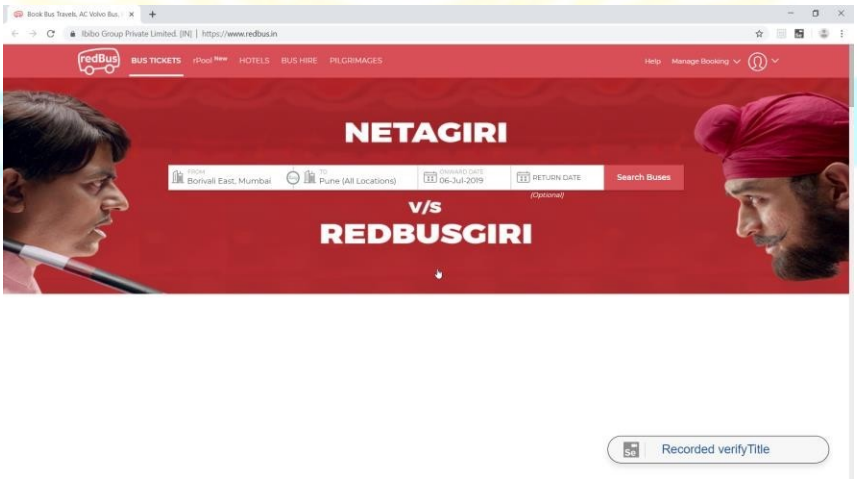
- A **new tab** will be opened, **maximize** that tab(this tab is your working tab forrecording):



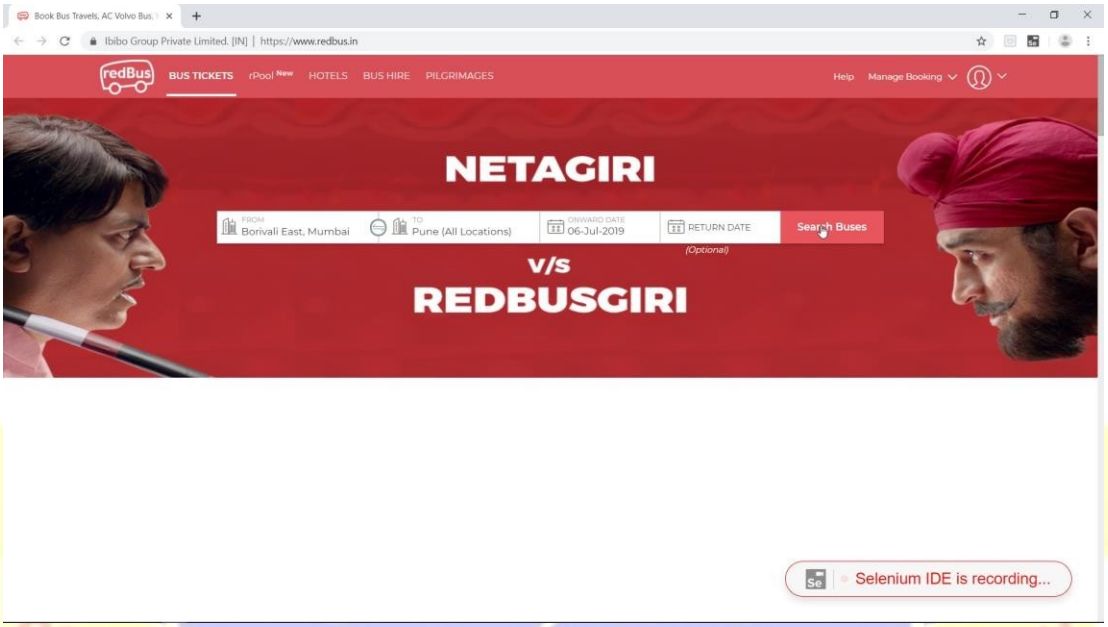
- Now we'll make use of command **verifyTitle**.
(This command verifies title of the webpage and keeps on continuing the execution.)So **anywhere on the window**, do **Right Click>** select **Selenium IDE>** click **Verify Title**:



- **NOTE THAT** each command is recorded:

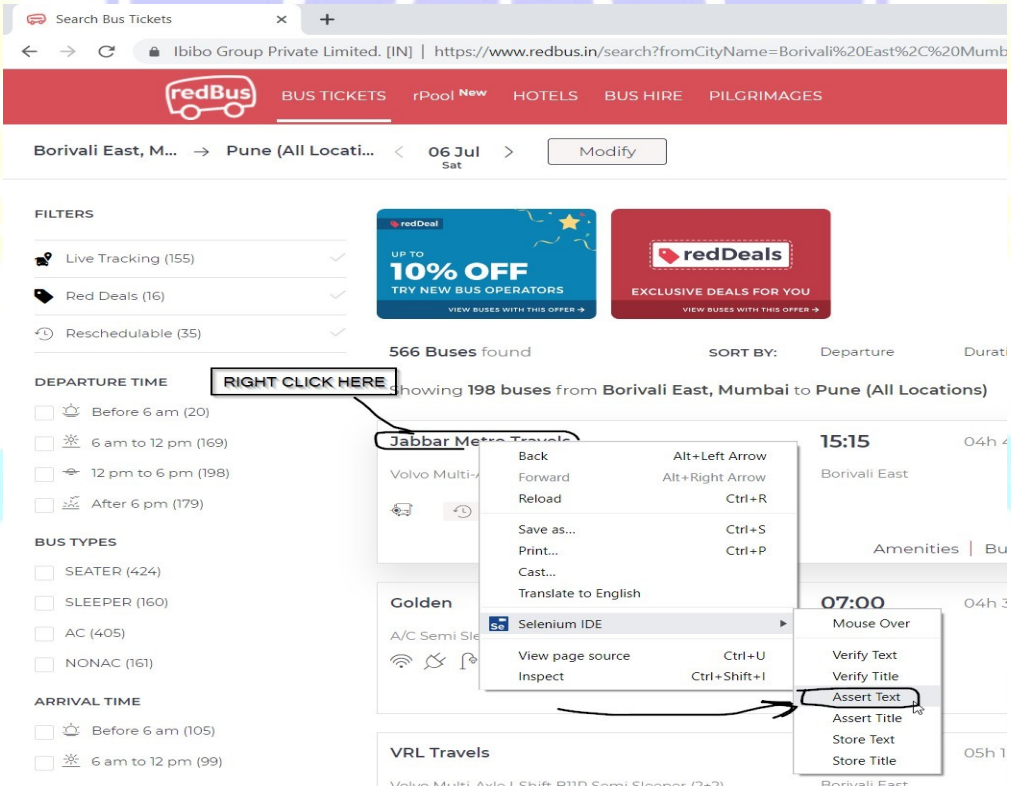


- Now after filling the FROM, TO, ONWARD DATE, we proceed further by clicking on **Search Buses:** (make sure not to do unnecessary clicks)

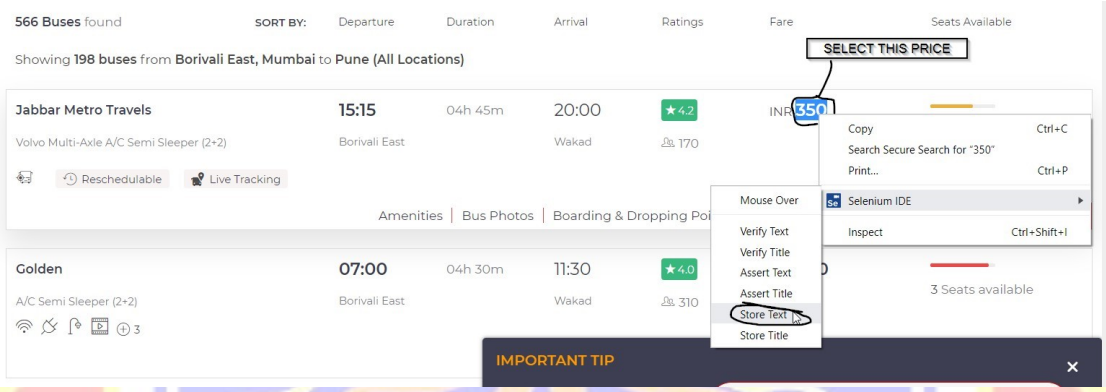


- Now we'll make use of command **assertText**.
  (This command verifies particular text of the webpage and stops the execution of testcase if the match isn't found.)
  So, we want to make sure that our automated testcase chooses our specified BusProvider.
  To ensure that, here we do **Right Click over the name of the bus provider>**

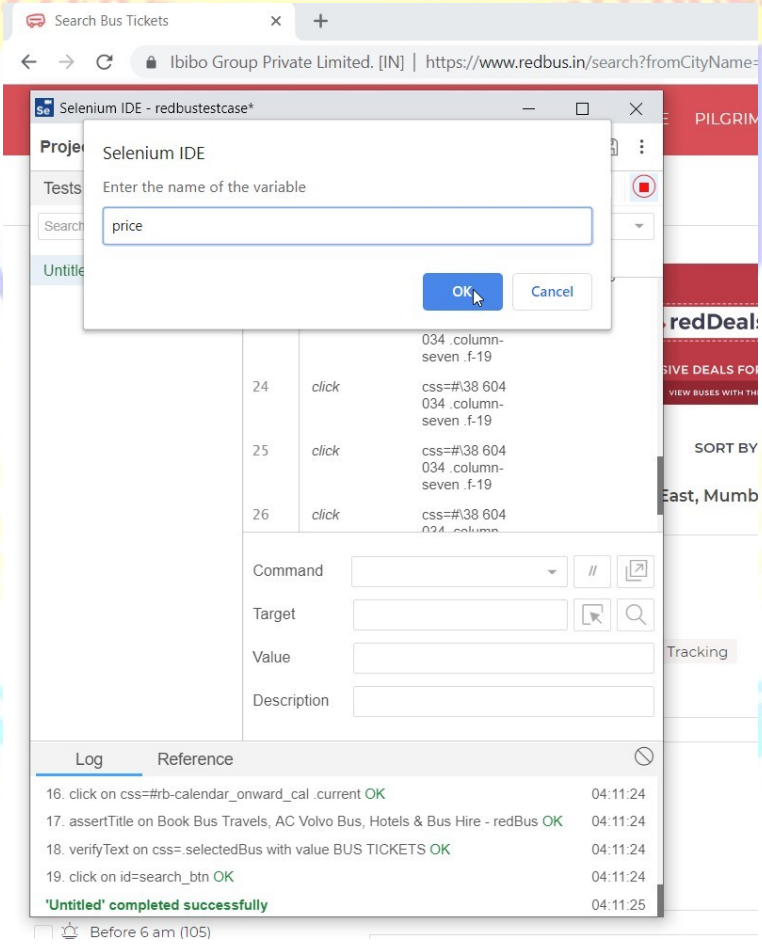
  then select **Selenium IDE>** and click **Assert Text** :



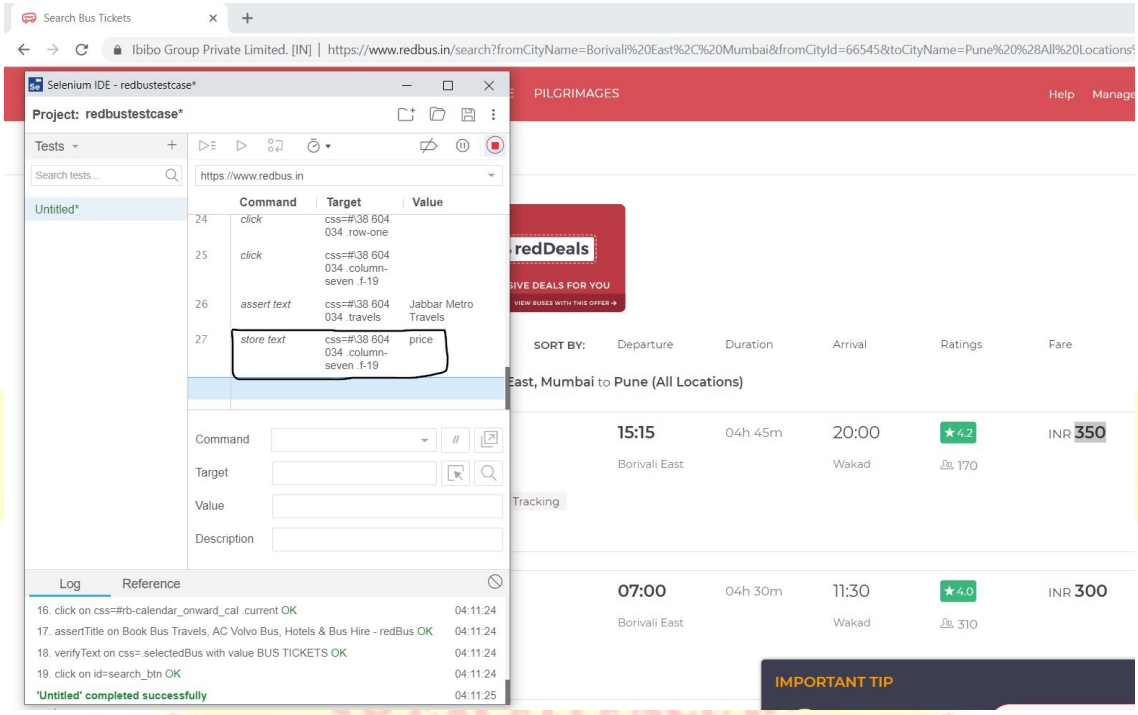
- Now we'll make use of command **storeText**.
  (This command stores the text value of page element into variable for future use.)
  Here we'll store the price of the ticket.
  So, we **select the price> right click** over it> select **Selenium IDE>** click **Store Text :**



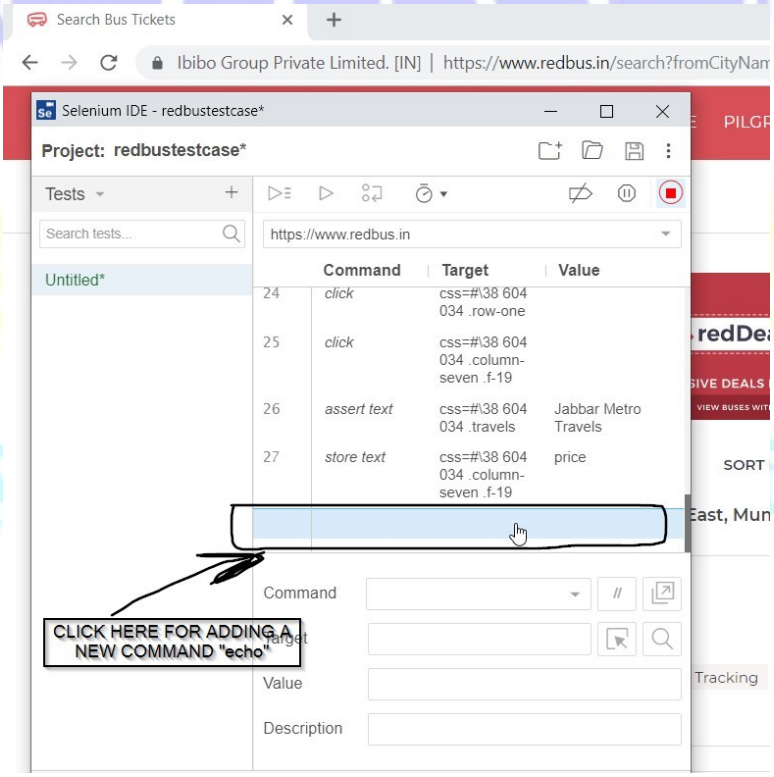
- Now Selenium IDE will prompt us to **enter the name of the variable** in which thatprice will be stored:



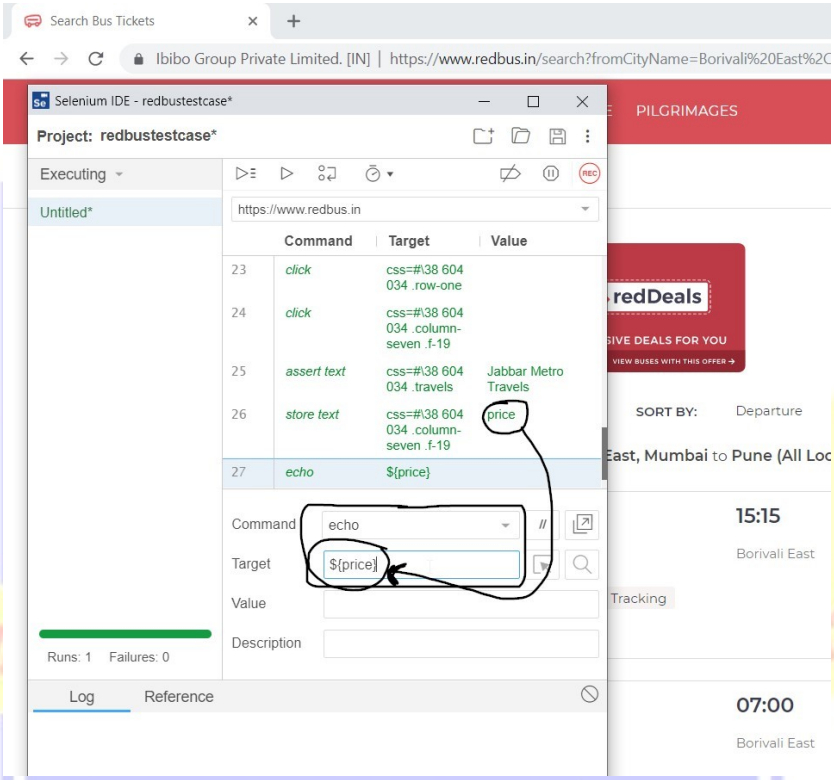
- After clicking OK, we can see how the target is automatically mapped to the cssattribute of the page, and our price is stored:



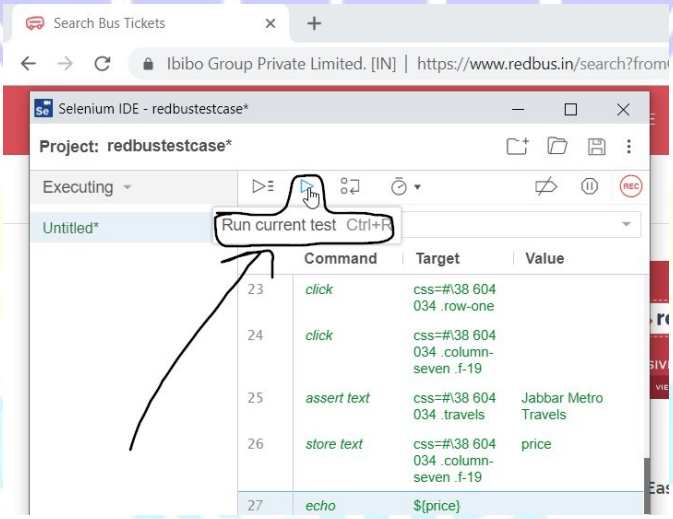
- Now we'll make use of command **echo.** (This command is used to display text(comments, notes) and/or the stored value ofvariables in the log area of Selenium IDE.) Here we'll display the price of the ticket from the variable "price" that we stored during the use of storeText command. To create a new command, we just click onto the next line and enter our command:



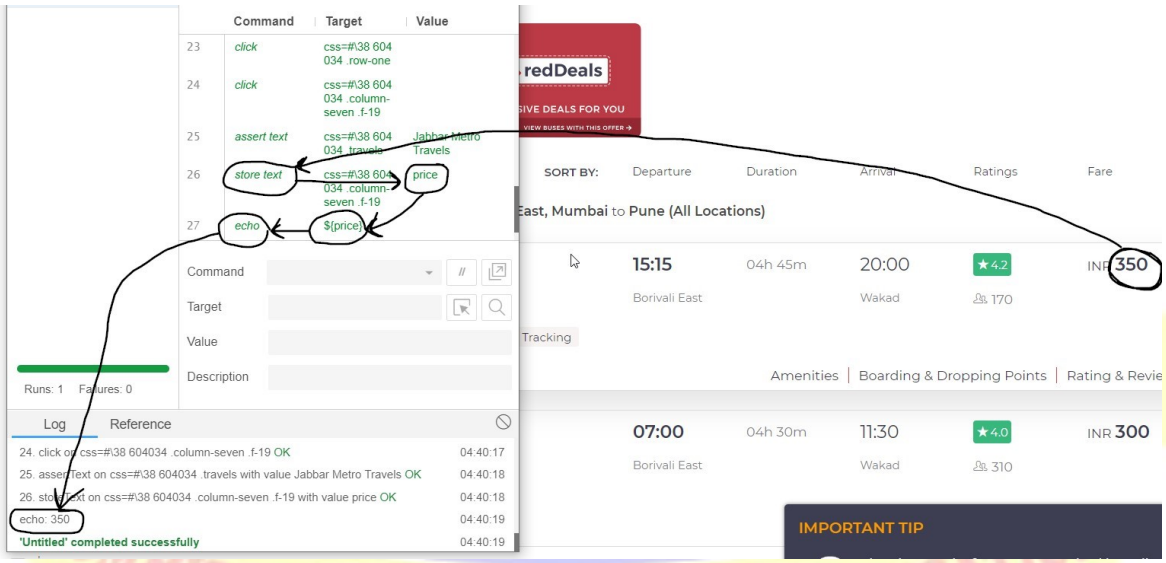
- We'll enter the Command name **echo**, & Value **${price}** because we are calling the created variable "price" from this command:



- Now we'll **Run current test**:



- After the test gets completed, we can see the **echo: 350** in the log area of SeleniumIDE showing the output of the echo command. Here we can see how 350 is stored in variable "price" by storeText command, and that variable is then called by the echo command, which is then returning the valuefetched from the page:



3) Finish!

(Assert and verify commands are both useful for verifying condition match or not. The differenceis that verify command will verify the condition and if it's not match, it will give error message in Log area and the macro(testcase) continues to run. With the assert command, if the condition does not match then it will stop remaining macro(testcase) execution in the selenium IDE software testing tool.)

Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) What is the purpose of creating a test suite in Selenium IDE?

2) How do you record a test case in Selenium IDE for clicking a link or button?

3) How would you use Selenium IDE to fill out a form on a website?

4) How can you verify that a certain piece of content is present on a webpage using Selenium IDE?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 3 : Installing Selenium Server (Selenium RC)

Selenium Remote Control (RC), also known as Selenium Server, was one of the original components of the Selenium suite that allowed automation of web application testing. It enabled running tests written in different programming languages across various browsers. Though it has largely been replaced by Selenium WebDriver, understanding Selenium RC is useful for historical context and legacy projects.

1. Overview of Selenium RC

- Selenium RC is a server that allows users to write test scripts in various programming languages (like Java, Python, Ruby, etc.) and run them on different web browsers.
- It works by using a client-server architecture, where the Selenium server acts as a proxy for web browser requests.
- Selenium RC interacts with the browser through JavaScript injection, enabling control over the browser's DOM (Document Object Model) for test automation.

3. Architecture of Selenium RC

- Selenium Server: Acts as a middle layer between the test scripts and the web browser.
    - It receives commands from the test script and converts them into browser-specific actions.
    - The server uses JavaScript to execute browser actions.
- Client Libraries: The Selenium client libraries are used to write test scripts in different programming languages.
- Browser Drivers: Selenium RC interacts with different web browsers using specific drivers that understand how to communicate with the browser.

4. Steps to Install Selenium RC (Selenium Server)

- Step 1: Download Selenium Server
    - Download the Selenium Server standalone JAR file from the official Selenium website. The file is usually named something like selenium-server-standalone-x.xx.x.jar.
- Step 2: Configure the Environment
    - Ensure that Java Runtime Environment (JRE) is installed on your machine, as Selenium Server requires Java to run.
    - Set up environment variables for Java if necessary (e.g., add Java to the system's PATH).
- Step 3: Start the Selenium Server
    - Open a command prompt or terminal window.
    - Navigate to the directory where the Selenium Server JAR file is located.

- Run the following command to start the server:

Copy code

```
java -jar selenium-server-standalone-x.xx.x.jar
```

- The server will start and listen on a specified port (default is 4444).

- Step 4: Configure the Test Script
  - Set up your test scripts to communicate with the Selenium Server by specifying the server's IP address and port number.
  - Write test cases using the Selenium client libraries for the programming language of your choice.

5. Advantages of Selenium RC

- Multi-browser Compatibility: It supports testing across a wide range of browsers.
- Programming Language Flexibility: Allows test scripts to be written in different languages like Java, C#, Python, Ruby, Perl, etc.
- Remote Execution Capability: Tests can be executed on a different machine or server, enabling distributed testing setups.

6. Limitations of Selenium RC

- Complexity: Setting up and maintaining a Selenium RC environment is more complex compared to Selenium WebDriver.
- Slower Execution: Tests run slower because Selenium RC relies on JavaScript to interact with the browser.
- Obsolescence: Selenium RC is now deprecated and no longer actively maintained. Selenium WebDriver replaced it because it provides a more direct and efficient way of controlling browsers.
- Limited Browser Support: Modern browsers and web standards may not be fully supported.

7. Transition from Selenium RC to WebDriver

- Selenium WebDriver was introduced to overcome the limitations of Selenium RC by offering more direct browser control, bypassing the need for JavaScript injection.
- WebDriver has better integration with modern browsers and supports advanced features like handling alerts, frames, and file uploads more effectively.

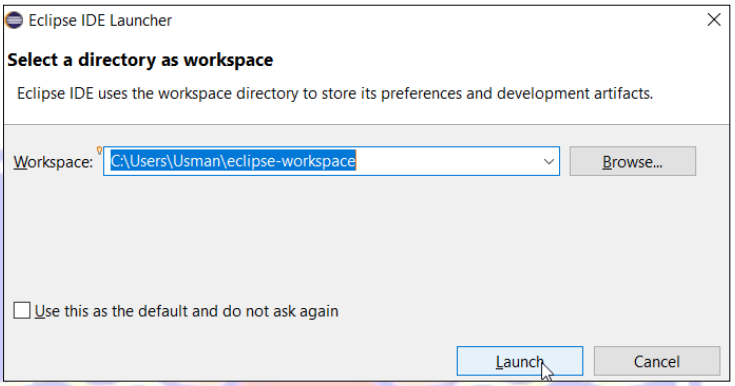
Practical 3 : Install Selenium Server (Selenium RC) and demonstrate its usage by executing a script in Java to automate browser actions.

PRE-REQUISITES(*here are w.r.t. Windows 10(64 bit), so choose accordingly w.r.t. your specs)**:**

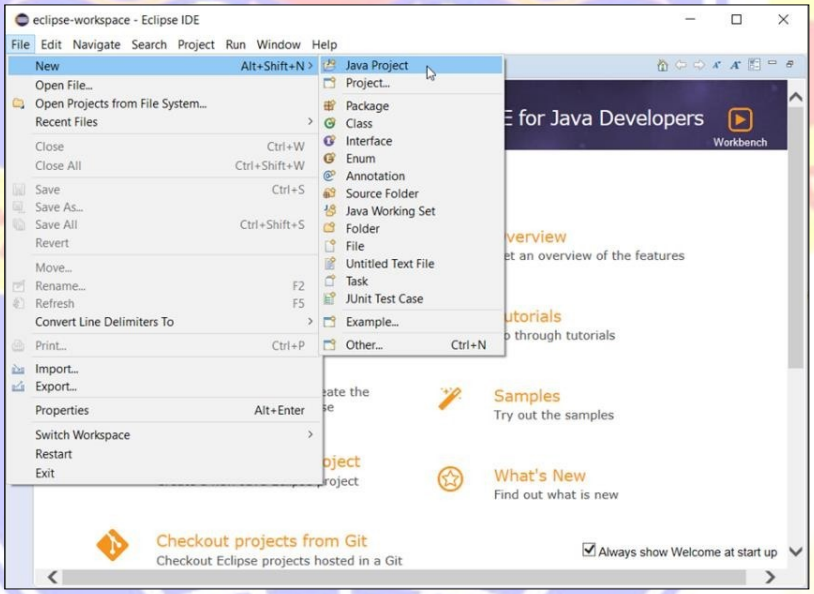
1) To Download "**JDK**":
   - **Visit** https://www.oracle.com/technetwork/java/javase/downloads/jdk12-downloads-5295953.html
   - **Download** this file "jdk-12.0.2_windows-x64_bin.exe" and **install it**.
2) To Download "**Eclipse IDE**":
   - **Visit** https://www.eclipse.org/downloads/download.php?file=/oomph/epp/2019-06/R/eclipse-inst-win64.exe
   - Click "**Download**".
   - Installation:
     - **Open application**.(click OK if errors occur like "could not find java.dll" & "couldnot find Java Runtime Environment SE")
     - Select "**Eclipse IDE for Java Developers**".
     - It will automatically locate the JDK. Choose path, and click "**Install**".
   - After installation, click "**Launch**" or open Eclipse from START menu in Windows.
3) To Download "**Selenium *Server Driver* and *Client Driver*(JAR files)**":
   a) For "Selenium *Server Driver*":
      - **Visit** https://www.seleniumhq.org/download/
      - Under section "**Selenium Standalone Server**", click download version "**3.141.59**"
      - You'll get the executable jar file(selenium-server-standalone-3.141.59)
   b) For "Selenium *Client Driver*":
      - **Visit** https://www.seleniumhq.org/download/
      - Under section "**Selenium Client & WebDriver Language Bindings**", downloadthe "**3.141.59**" version of Java.
      - Extract the file and you'll see two jar files. From them, we'll be using thisexecutable jar file(client-combined-3.141.59)
4) To Download "**Gecko Driver**":
   - **Visit** https://github.com/mozilla/geckodriver/releases
   - Under section "**Assets**", download "geckodriver-v0.24.0-win64.zip" file.
   - You'll get the application file "geckodriver".

STEPS:

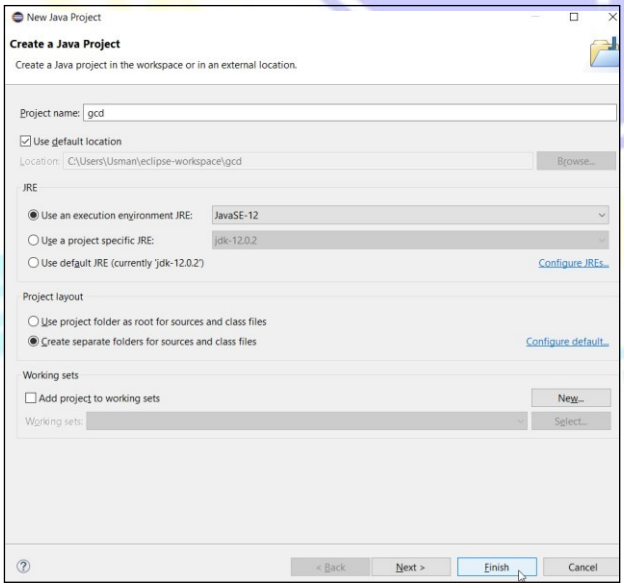
1) **Open Eclipse**. Select your workspace directory. Click **Launch:**

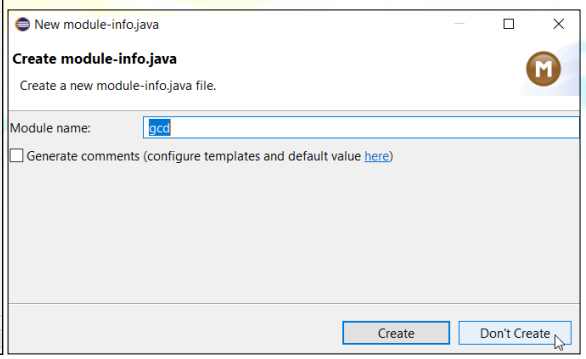


2) **Create** a Project(**File > New > Java Project**):



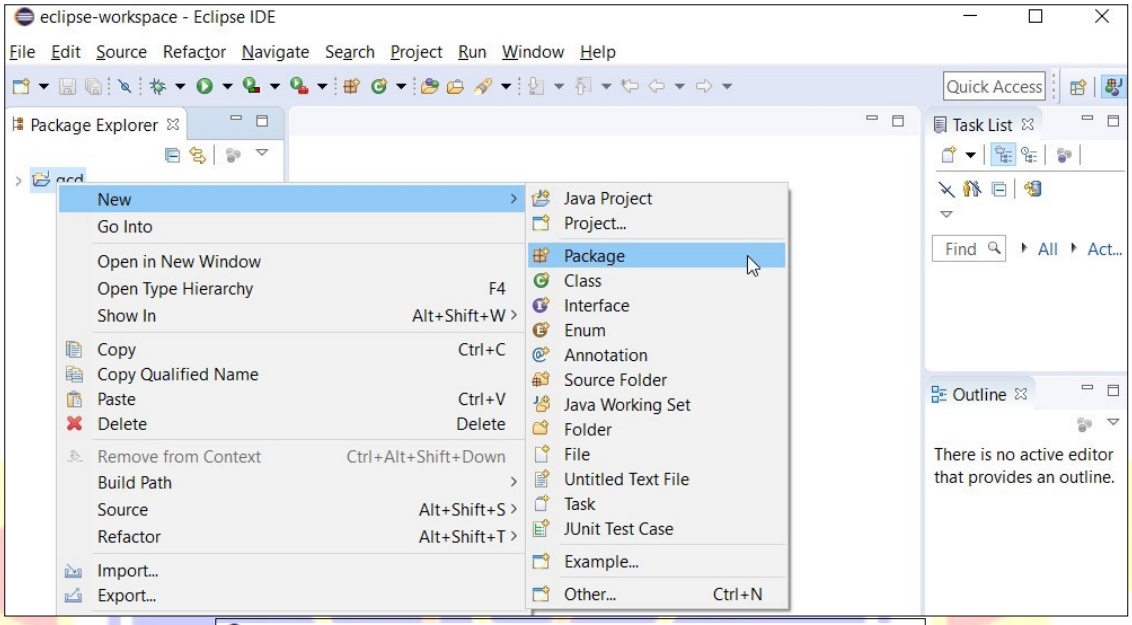
3) **Name the project** as "gcd" **>** click **Finish >** click **Don't Create** module:

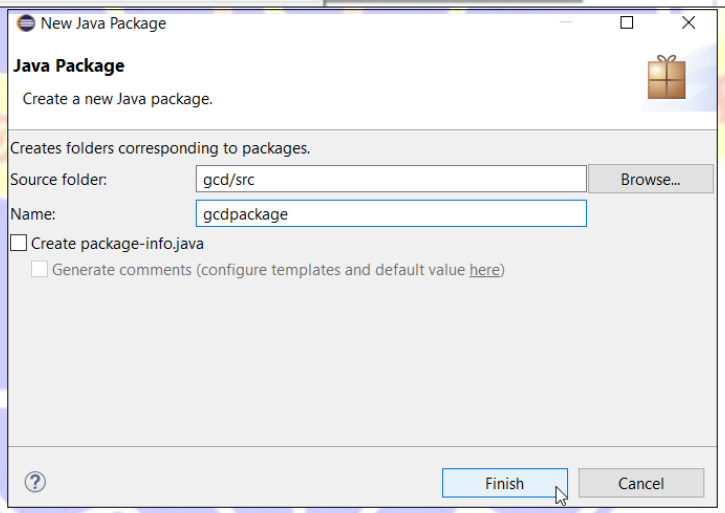




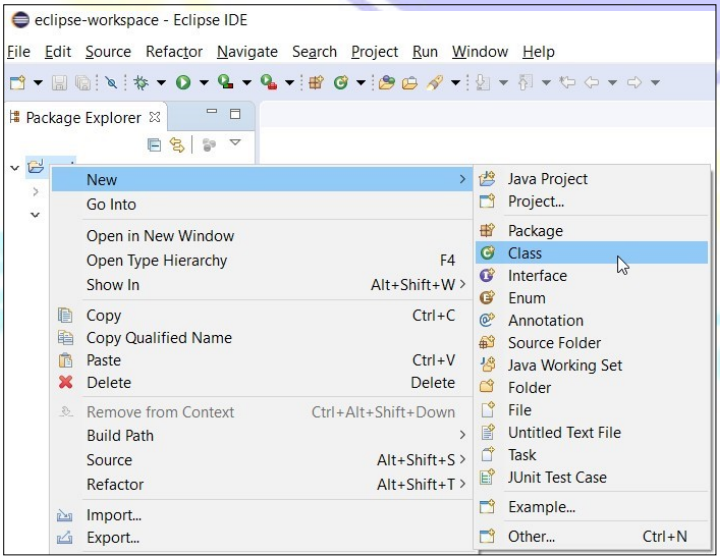
4) **Close the "Welcome" tab**.

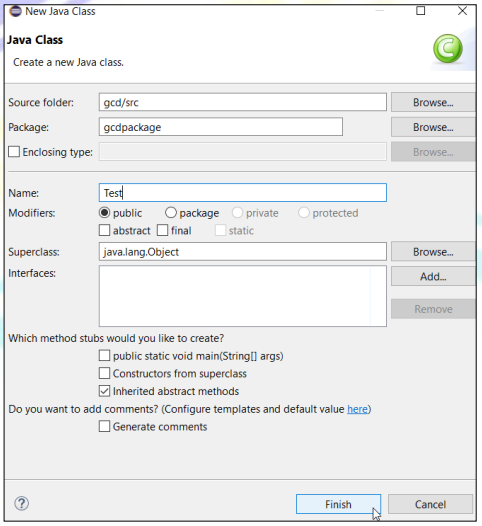
5) **Create** a Package(**right-click on Project Name > New > Package > Name** it **> Finish**):





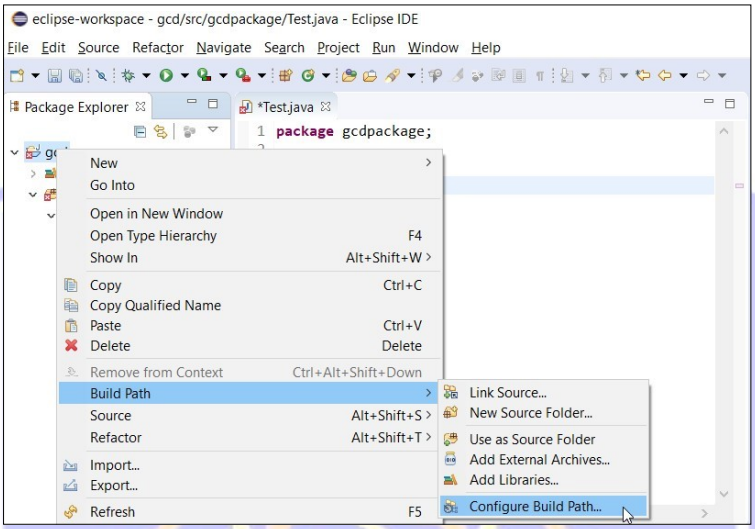
6) **Create** a Class(**right-click on Project Name > New > Class > Name** it **> Finish**):



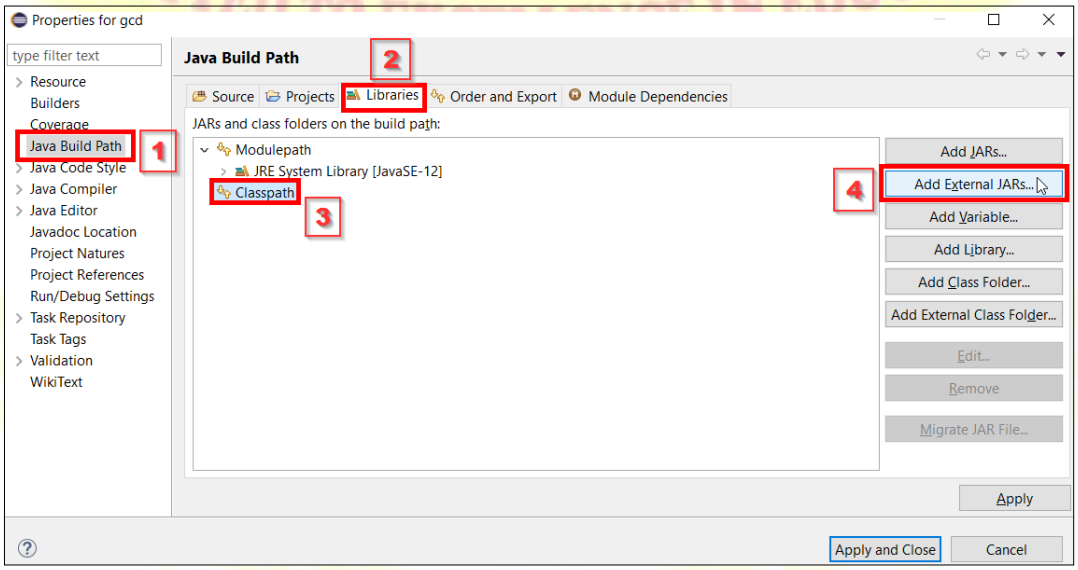


7) Adding "**Selenium *Server Driver* and *Client Driver*(JAR files)**" in Eclipse IDE:

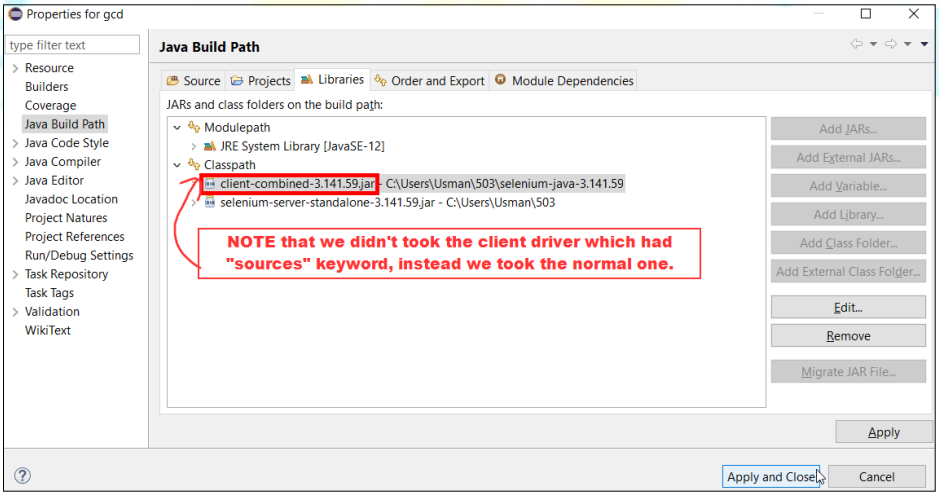
- **right-click on Project Name > Build Path > Configure Build Path…**



- now go under: **Java Build Path > Libraries > Classpath >** click **Add External JARs…**



- Browse and **add JAR files >** click **Apply and Close :**



1) **Creating a link for HTML file**(wherein calculation part is present):
(NOTE that this file will be run by the '***script*** *in JAVA'(which we'll create later)*)

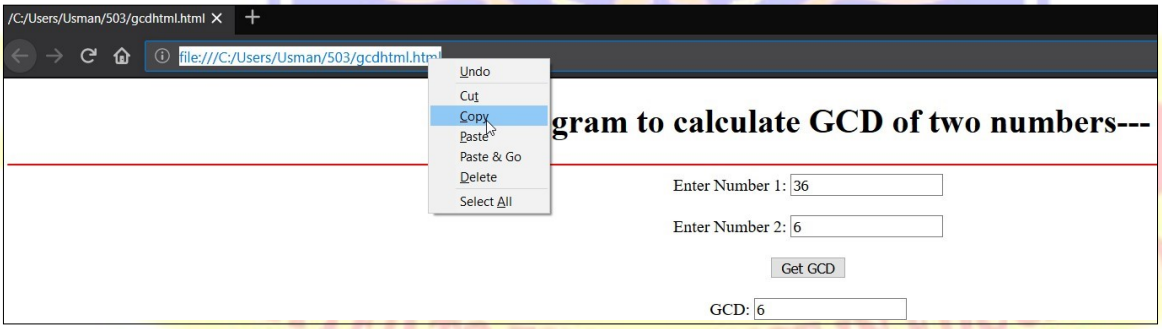
- **Create a Notepad file** with the following code and save it as "*gcdhtml.html*":

---(gcdhtml.html)---

```
<html>
<head>

<script type="text/javascript">
Function gcd()
        {
                var x,y;
                x=parseInt(document.myform.n1.value);
                y=parseInt(document.myform.n2.value);
                while(x!=y)
                {
                        If(x>y){x=x-y;}
                        Else{y=y-x;}
                }
                document.myform.result.value=x;
        }
</script>
</head>

<body>
<center>
        <h1>---Program to calculate GCD of two numbers---</h1>
        <hr color="red">

        <form name="myform">
                Enter Number 1: <input type="text" name="n1" value=""> <br> <br>
                Enter Number 2: <input type="text" name="n2" value=""> <br> <br>
```

```
<input type="button" name="btn" value="Get GCD" on Click="gcd()"><br><br>
GCD: <input type="text" name="result" value="">

</form>

</center>

</body>

</html>
```

- Close the file. Then right-click > Open with > Firefox Browser
  - **Copy URL** from the webpage:



8) Creating the ***script*** *in JAVA*:
(NOTE that this **script** will be run by Eclipse IDE)
(In simple words, it's like we are
-ordering Eclipse to run a script or to do a job
-of opening the HTML file
-and putting the values in the textboxes with the help of Selenium Drivers
-and to show the result.
-Hence automating the work in browser)

- Now we'll put the path of "geckodriver" in a String ***driverPath***
- And we'll **paste the copied URL in the *.get()* method** of the WebDriver class

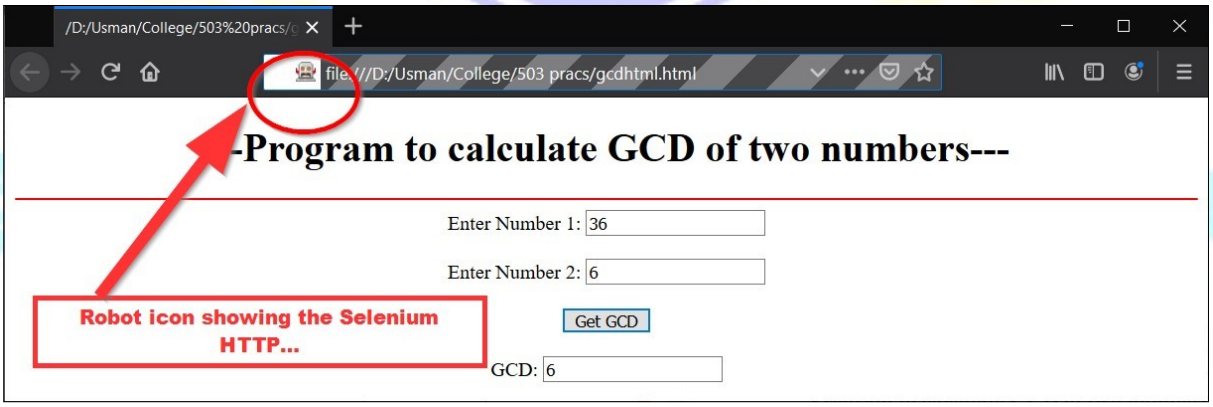
---(Test.java)---

```
package gcdpackage;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import
org.openqa.selenium.firefox.FirefoxDrive
r;import
org.openqa.selenium.firefox.FirefoxOptio
ns;import
org.openqa.selenium.firefox.FirefoxProfi
le;
```

```
public class Test {

	static String driverPath = "C:\\Users\\Usman\\503\\geckodriver.exe";

	public static void main(String[] args) {
		System.setProperty("webdriver.gecko.driver",driverPath);

		//DesiredCapabilities capabilities = DesiredCapabilities.firefox();

		//capabilities.setCapability("marionette",true);

		//ProfilesIni allProfiles = new
		ProfilesIni(); FirefoxProfile fp = new
		FirefoxProfile();
		fp.setPreference(FirefoxProfile.PORT_PREFER
		ENCE,"7055");FirefoxOptions options = new
		FirefoxOptions(); options.setProfile(fp);

		WebDriver driver=new FirefoxDriver(options);//(capabilities);

		//WebDriver driver=new ChromeDriver();
		driver.get("file:///D:/Usman/College/503%20pracs/gcdhtml.html");
		driver.manage().window().maximize();
		driver.findElement(By.name("n1")).sendKeys("36");
		driver.findElement(By.name("n2")).sendKeys("6");
		driver.findElement(By.name("btn")).click();

		String
result=driver.findElement(By.name("result")).getAttribute("name
=result");

		System.out.println("GCD="+result);

	}

}
```

9) **Run** the file from Eclipse IDE:

- **OUTPUT:**



10) **Finish!**

What is Gecko Driver:

-a WebBrowser engine inbuilt within Mozilla Firefox browser.

-acts as a proxy between Web Driver enabled clients(Eclipse, NetBeans, etc.) and Mozilla Firefox browser *or simply* acts as a link between Selenium Web Driver tests and Mozilla Firefox browser.

What is Selenium Remote Control (RC):

-a test tool that allows you to write automated web application UI tests in any programminglanguage against any HTTP website using any mainstream JavaScript-enabled browser.

-Selenium RC comes in two parts.

1) A server which automatically launches and kills browsers, and acts as a HTTP proxy for webrequests from them.
2) Client libraries for your favourite computer language.

Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) What is Selenium RC, and how does it differ from Selenium WebDriver?

2) How do you install and start Selenium Server (Selenium RC)?

3) What is the purpose of writing a Java script to control the browser with Selenium RC?

4) How would you set up and run a basic Java Selenium RC test case?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 4 : Selenium Web Driver

Selenium WebDriver is a powerful tool for automating web browser interactions. It allows you to control a browser programmatically, simulating user actions such as clicking, typing, and navigating through web pages. Here's an overview of Selenium WebDriver, including its setup and basic use cases:

1. What is Selenium WebDriver?

- Selenium WebDriver is a part of the Selenium suite, which provides tools for web browser automation.
- It allows for cross-browser testing, meaning you can run the same tests on different browsers like Chrome, Firefox, Safari, and Edge.
- WebDriver interacts directly with the browser's native functionality, providing a more accurate simulation of user actions than older tools like Selenium RC.

2. How Does Selenium WebDriver Work?

- Selenium WebDriver uses browser-specific drivers to interact with the browsers. For example, ChromeDriver is used for Chrome, and GeckoDriver is used for Firefox.
- It communicates with the browser through a web browser driver, which translates the commands in your test script into actions on the browser.

3.**Advantages of Selenium WebDriver**

- Cross-Browser Compatibility: Can run tests on different browsers.
- Support for Multiple Programming Languages: Allows test scripts to be written in Python, Java, C#, Ruby, etc.
- Direct Browser Interaction: Communicates with the browser directly, providing a more accurate automation experience.
- Flexibility in Locating Elements: Offers different methods to locate and interact with elements on a webpage.

4. **Limitations of Selenium WebDriver**

- Not Suitable for Non-Web-Based Applications: Selenium only automates web browsers.
- Handling Complex Actions: Tasks like captcha handling and visual validation are challenging to automate with Selenium alone.
- Browser Dependencies: Requires keeping WebDriver versions in sync with browser versions.

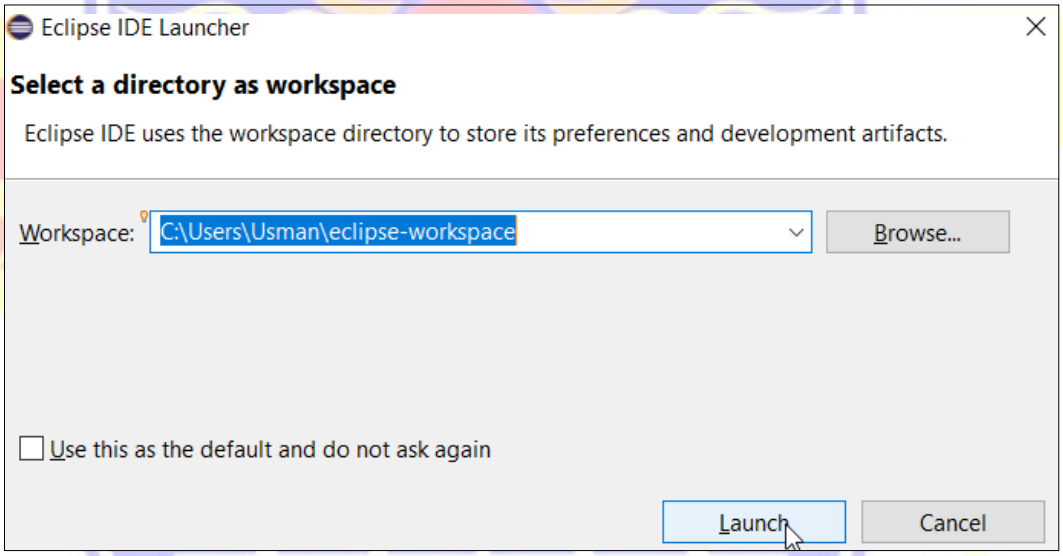
Practical 4: Write a program using Selenium WebDriver to update 10 student records in an Excel file. Perform data manipulation and verification.

PRE-REQUISITES:

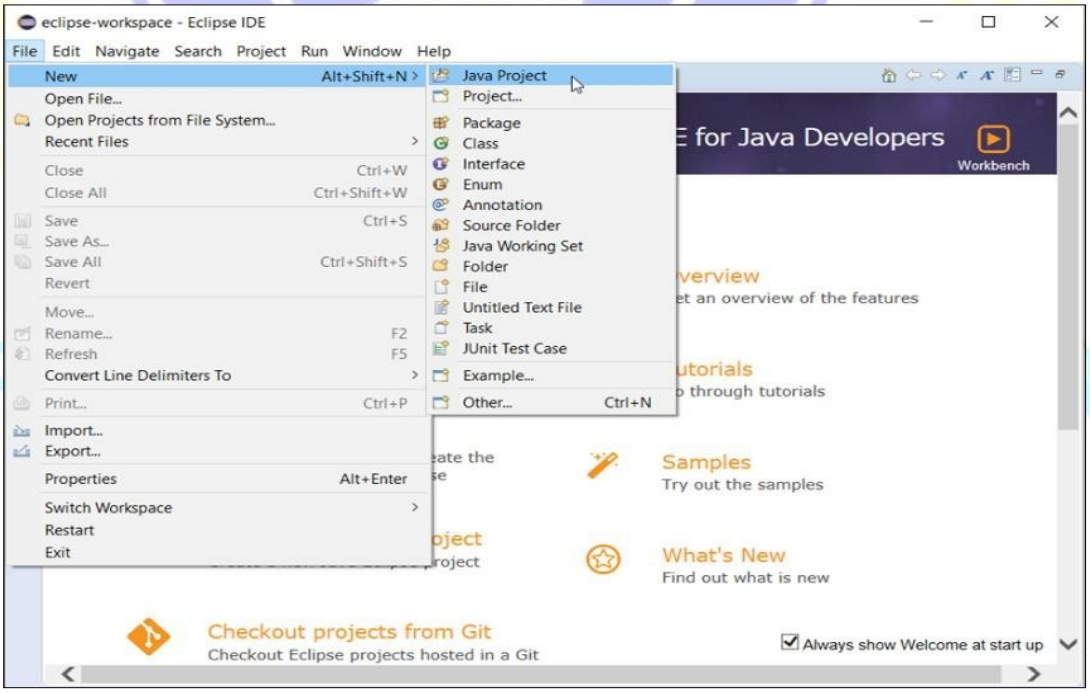
1) Check that you have **Eclipse IDE**.
2) To Download "**JXL.JAR**":
   - **Visit** http://www.java2s.com/Code/Jar/j/Downloadjxl26jar.htm
   - **Download** this file: "jxl/jxl-2.6.jar.zip(603 k)" and **extract it**.(you'll get the .jar file)

STEPS:

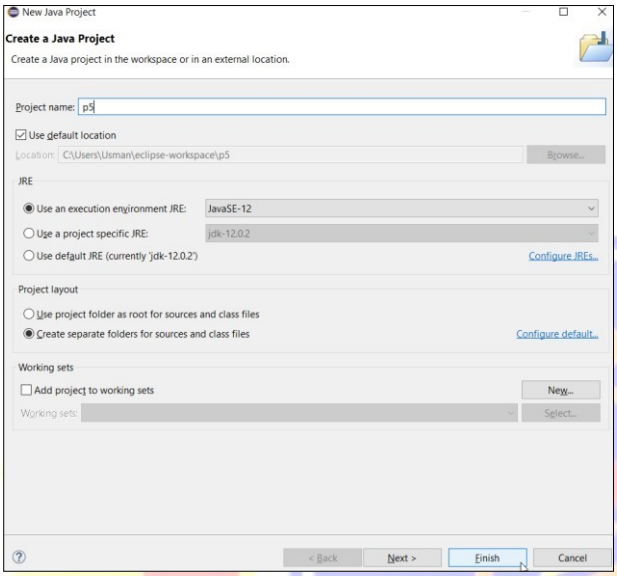
1) **Open Eclipse**. Select your workspace directory. Click **Launch:**

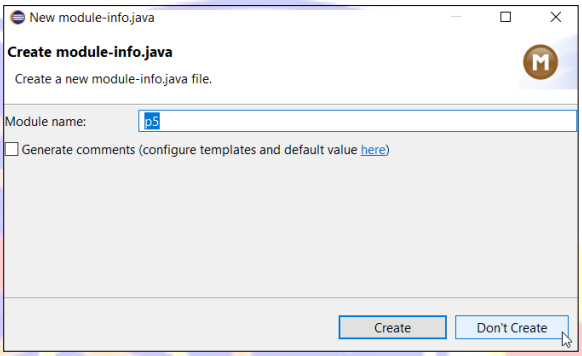


2) **Create** a Project(**File > New > Java Project**):



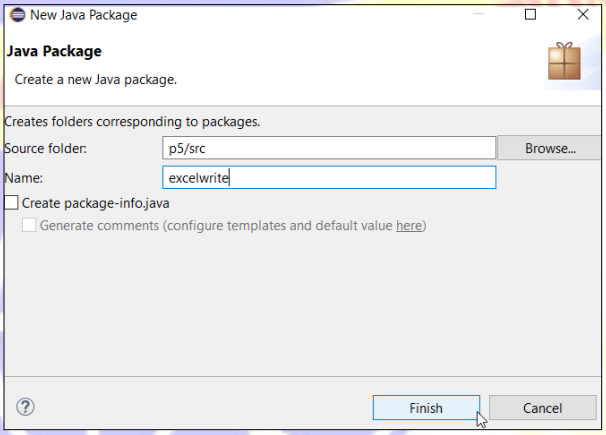
3) Name the project as "p5" **>** click **Finish >** click **Don't Create** module:

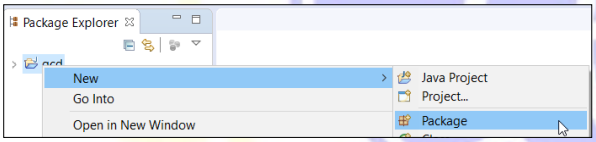




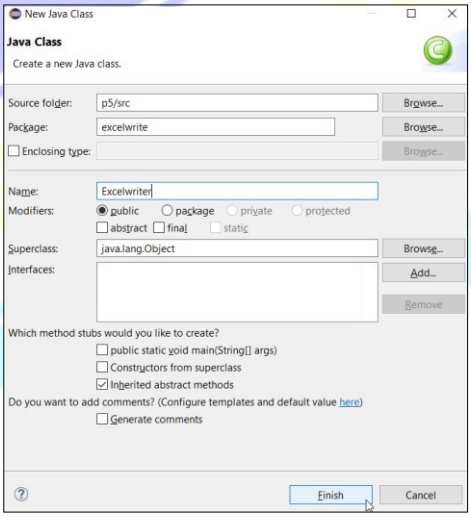
3) Close the "Welcome" tab.

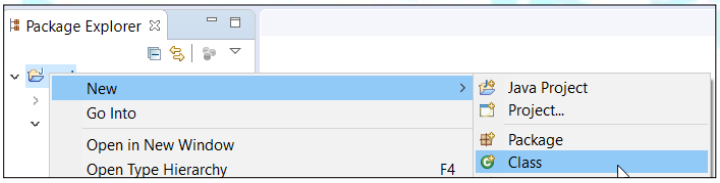
4) Create a Package(**right-click on Project Name > New > Package > Name** it **> Finish**):





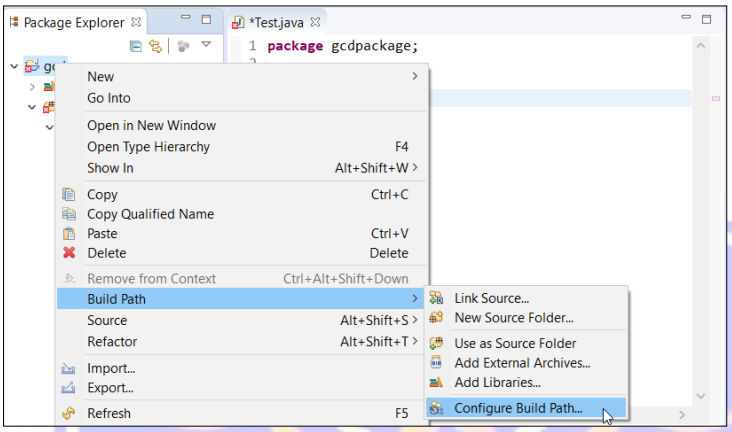
5) Create a Class(**right-click on Project Name > New > Class > Name** it **> Finish**):



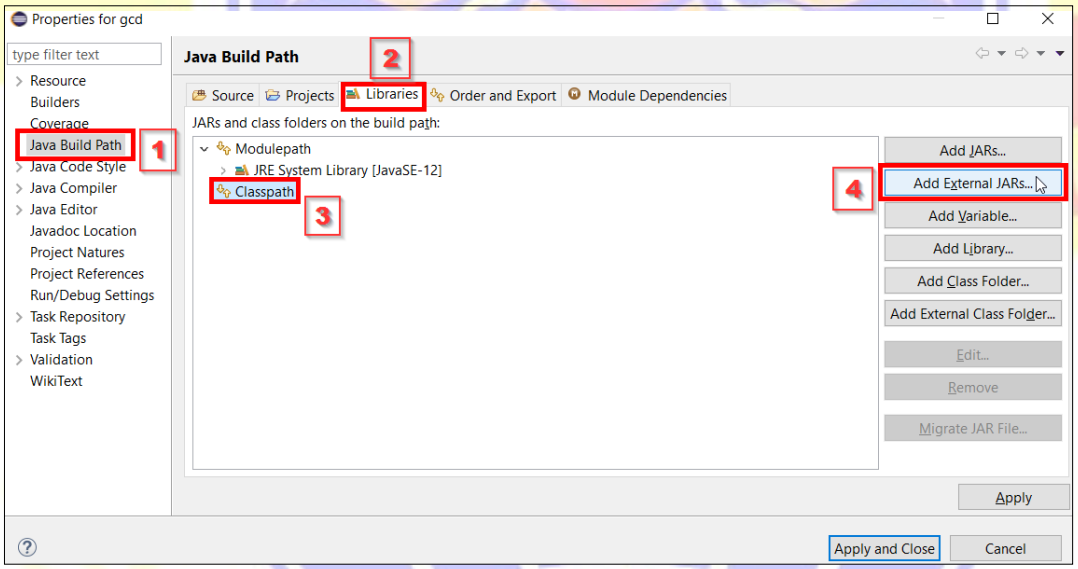


6) Adding "**JXL(JAR file)**" in Eclipse IDE:

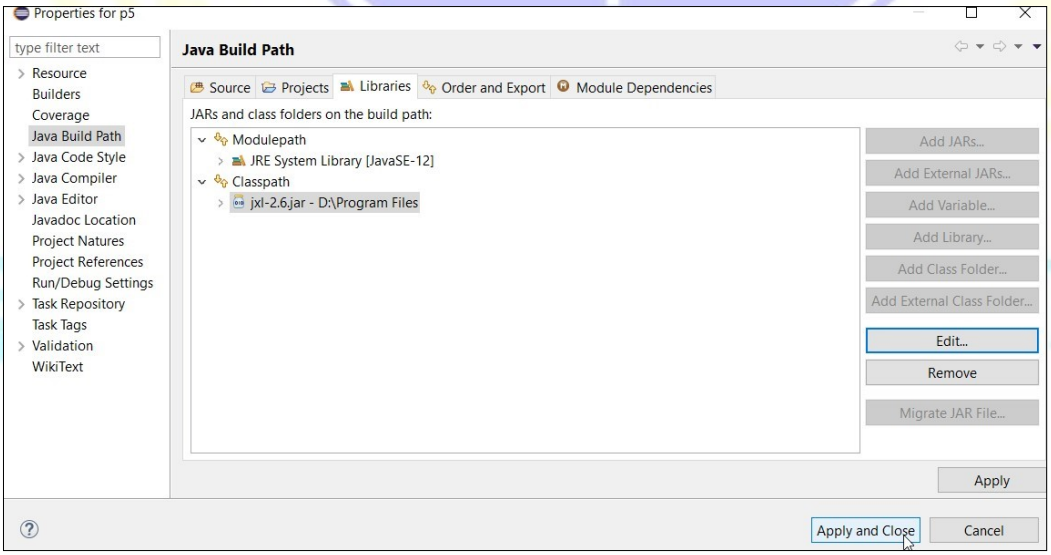
- **right-click on Project Name > Build Path > Configure Build Path…**



- now go under: **Java Build Path > Libraries > Classpath >** click **Add External JARs…**



- Browse and **add JAR file >** click **Apply and Close :**



7) Creating the ***script*** *in JAVA*:

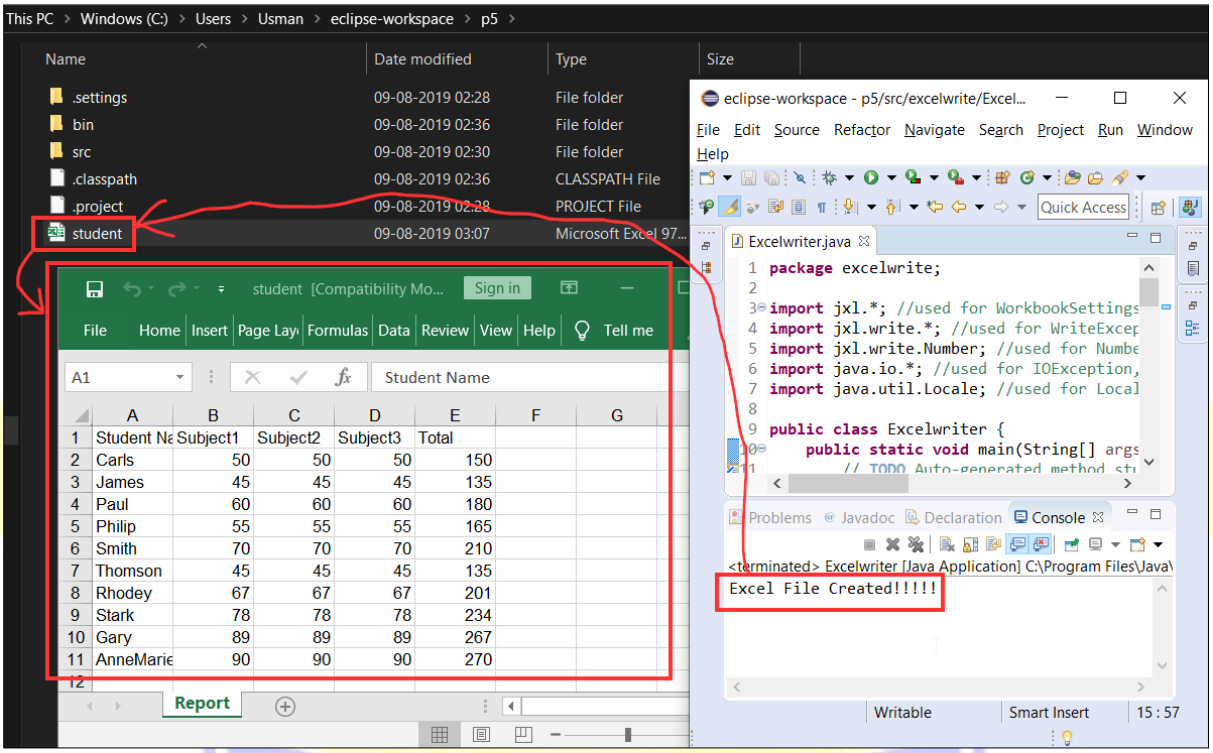
(NOTE that this **script** will be run by Eclipse IDE)
(In simple words, it's like we are
-ordering Eclipse to run a script or to do a job
-of creating and opening .xls file
-and putting the values in the cells with the help of jxl.jar
-and to show the result.
-Hence automating the work in a local system(PC)).

---(Excelwriter.java)---

```
package excelwrite;

import jxl.*; //used for WorkbookSettings,Workbook

import jxl.write.*; //used for WriteException,WritableWorkbook,WritableSheet,Label

import jxl.write.Number; //used for Number
import java.io.*; //used for IOException,File
import java.util.Locale; //used for Locale

public class Excelwriter {

	public static void main(String[] args) throws IOException,WriteException {

		// TODO Auto-generated method stub

		int r=0,c=0;

		String header[]={"Student
Name","Subject1","Subject2","Subject3","Total"};

		String
sname[]={"Carls","James","Paul","Philip","Smith","Thomson","Rhodey","Stark","Gary"

,"AnneMarie"};

		int marks[]={50,45,60,55,70,45,67,78,89,90,30};

		File file = new File("student.xls");
		WorkbookSettings wbSettings = new WorkbookSettings();
		wbSettings.setLocale(new Locale("en", "EN"));

wbSettings);		WritableWorkbook workbook = Workbook.createWorkbook(file,

		workbook.createSheet("Report",  0);
		WritableSheet excelSheet = workbook.getSheet(0);

		//creating header row

		for(r=0;r<1;r++) {

			for(c=0;c<header.length;c++) {

				Label l=new Label(c,r,header[c]);
				excelSheet.addCell(l);

			}

		}
```

```
//filling name in column1
for(r=1;r<=sname.length;r++) {
        for(c=0;c<1;c++) {
                Label l=new
                Label(c,r,sname[r-
                1]);
                excelSheet.addCell(l
                );
        }
}
//filling name in column2,3,4
for(r=1;r<=sname.length;r++) {
        for(c=1;c<4;c++) {
                Number num = new Number(c,
                r, marks[r-1]);
                excelSheet.addCell(num);
        }
}
//filling name in total
for(r=1;r<=sname.length;r++) {
        for(c=4;c<5;c++) {
                int total=marks[r-1]+marks[r-
                1]+marks[r-1];Number num =
                new Number(c, r, total);
                excelSheet.addCell(num);
        }
}
 workbook.write();
 workbook.close();
System.out.println("Excel File Created!!!!!");
    }
}
```

1) **Run** the file from Eclipse IDE:
   - **OUTPUT:**



2) **Finish!**

What is JXL.JAR:

-Java libraries are distributed as a “.jar” file

-jxl.jar is the library of JExcelApi , which is open source java API to read, write, and modify Excelspreadsheets dynamically. It contains all the compiled *.class files, associated metadata and resources that are used by the Java Excel API internally .

Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) What is Apache POI, and why is it used in this program?

2) How does Selenium WebDriver interact with the web elements in the student management system?

3) What steps would you take to verify if the student record was successfully updated?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 5: Selenium Web Driver

Selenium WebDriver is a powerful tool for automating web browser interactions. It allows you to control a browser programmatically, simulating user actions such as clicking, typing, and navigating through web pages. Here's an overview of Selenium WebDriver, including its setup and basic use cases:

1. What is Selenium WebDriver?

- Selenium WebDriver is a part of the Selenium suite, which provides tools for web browser automation.
- It allows for cross-browser testing, meaning you can run the same tests on different browsers like Chrome, Firefox, Safari, and Edge.
- WebDriver interacts directly with the browser's native functionality, providing a more accurate simulation of user actions than older tools like Selenium RC.

2. How Does Selenium WebDriver Work?

- Selenium WebDriver uses browser-specific drivers to interact with the browsers. For example, ChromeDriver is used for Chrome, and GeckoDriver is used for Firefox.
- It communicates with the browser through a web browser driver, which translates the commands in your test script into actions on the browser.

3.**Advantages of Selenium WebDriver**

- Cross-Browser Compatibility: Can run tests on different browsers.
- Support for Multiple Programming Languages: Allows test scripts to be written in Python, Java, C#, Ruby, etc.
- Direct Browser Interaction: Communicates with the browser directly, providing a more accurate automation experience.
- Flexibility in Locating Elements: Offers different methods to locate and interact with elements on a webpage.

4. **Limitations of Selenium WebDriver**

- Not Suitable for Non-Web-Based Applications: Selenium only automates web browsers.
- Handling Complex Actions: Tasks like captcha handling and visual validation are challenging to automate with Selenium alone.
- Browser Dependencies: Requires keeping WebDriver versions in sync with browser versions.

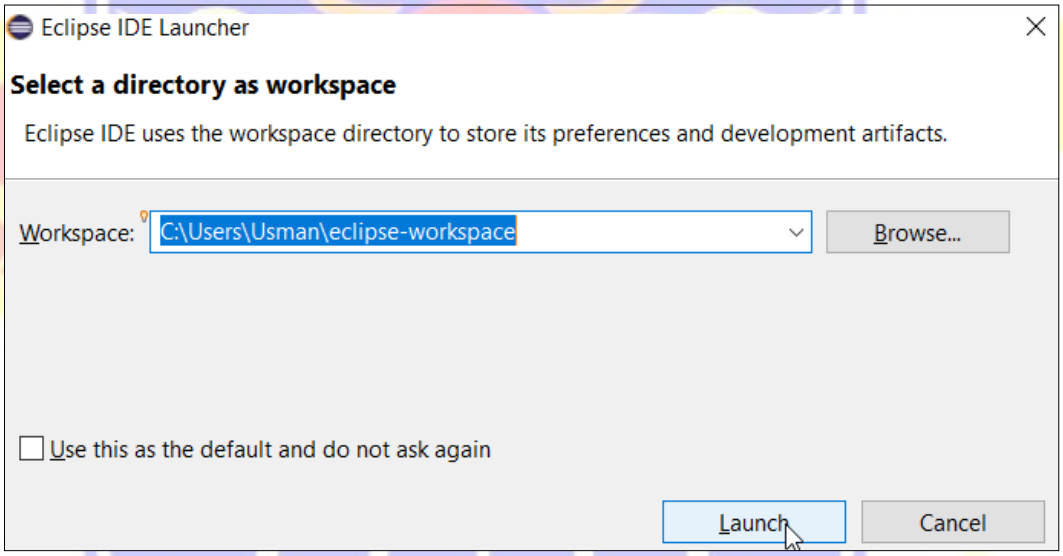
Practical 5: Write a program using Selenium WebDriver to select the number of students who have scored more than 60 in any one subject (or all subjects). Perform data extraction and analysis.

PRE-REQUISITES:

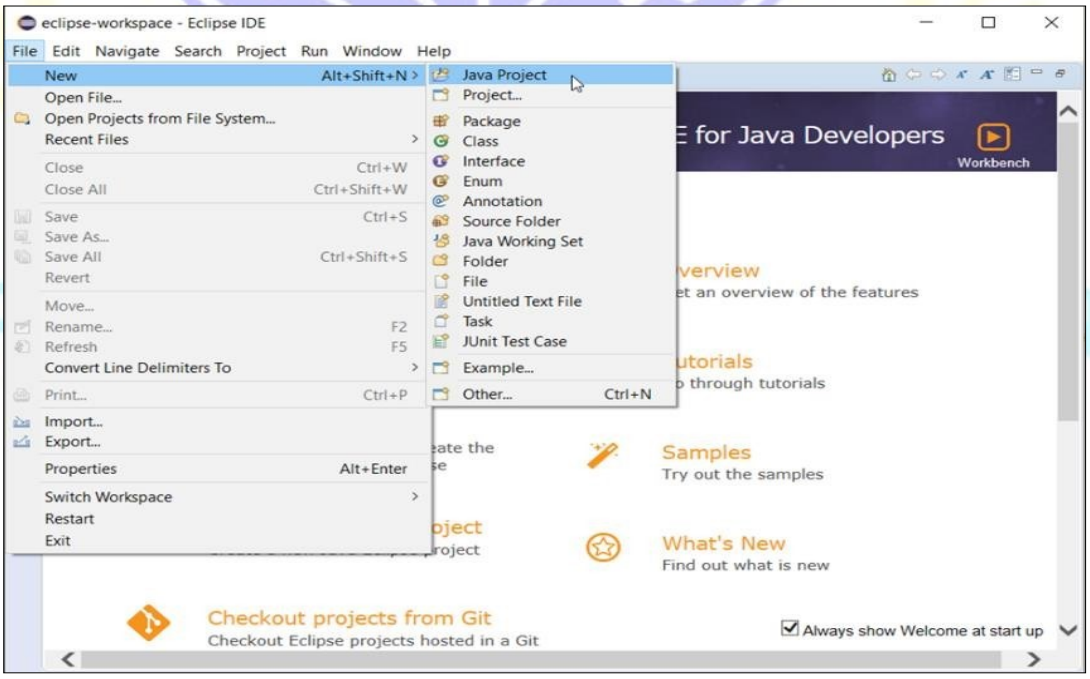
1) Check that you have **Eclipse IDE**.
2) Check that you have **JXL.JAR**.
3) Check that you've the ***Excel file****("student.xls")* that we'll be working on for reading data.

STEPS:

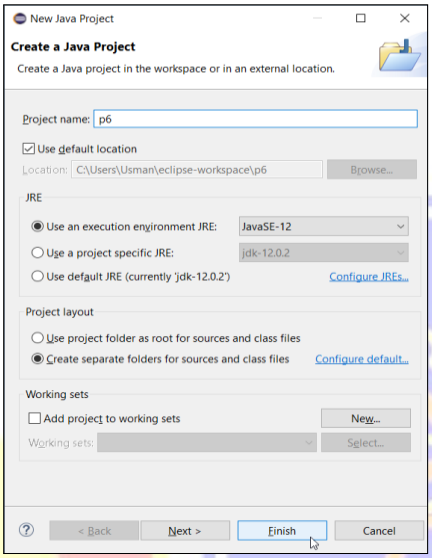
1) **Open Eclipse**. Select your workspace directory. Click **Launch:**

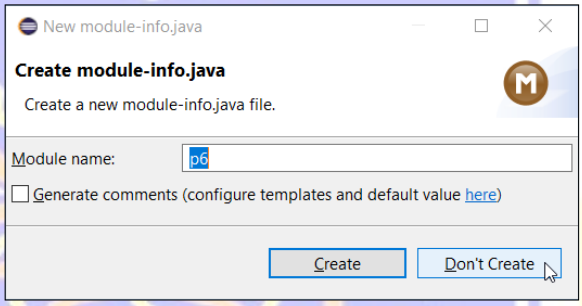


2) **Create** a Project(**File > New > Java Project**):



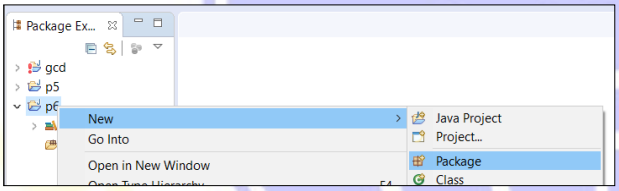
3) **Name the project** as "p6" > click **Finish** > click **Don't Create** module:

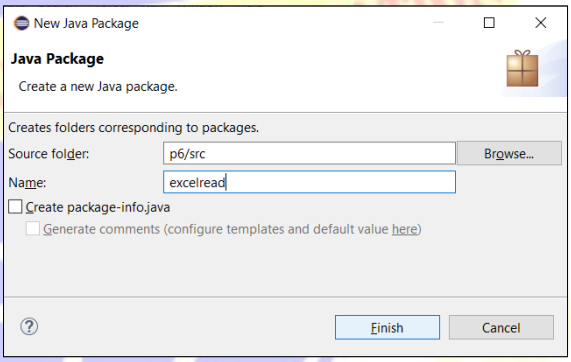




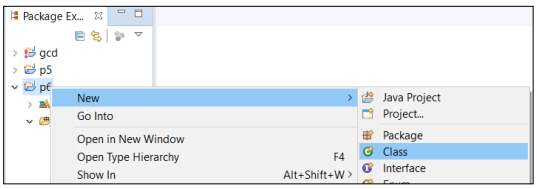
4) **Close the "Welcome" tab.**

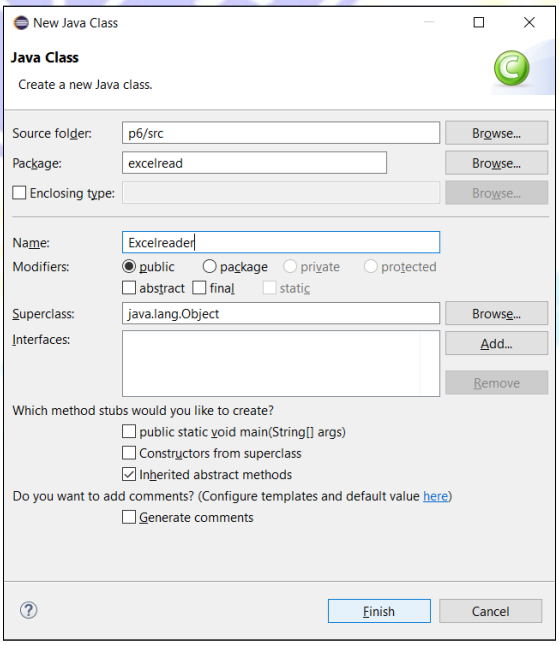
5) **Create** a Package(**right-click on Project Name > New > Package > Name** it **> Finish**):





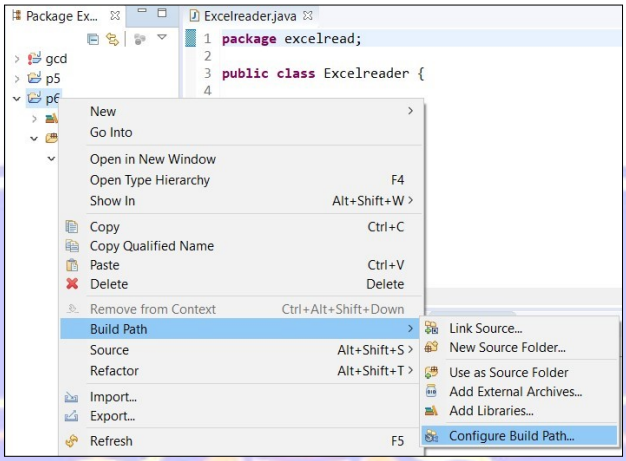
6) **Create** a Class(**right-click on Project Name > New > Class > Name** it **> Finish**):



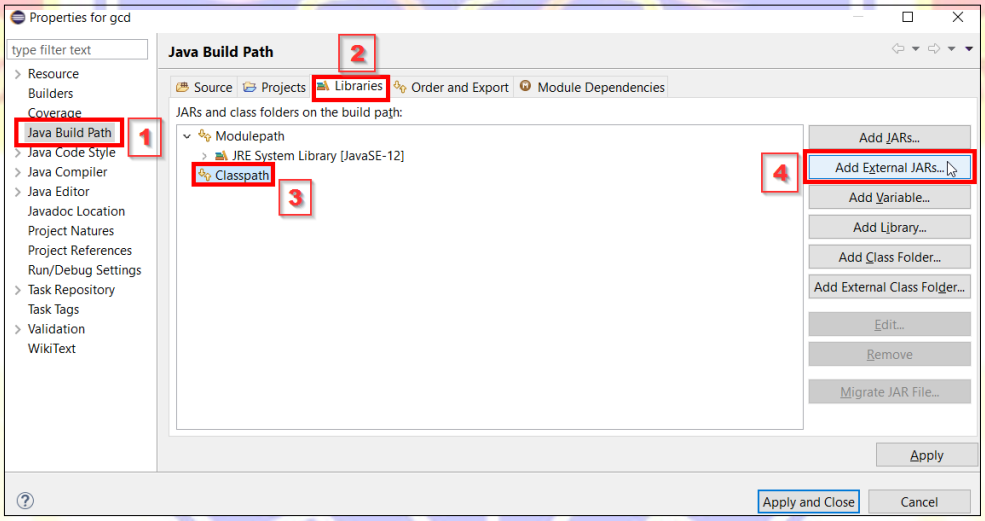


7) Adding "**JXL(JAR file)"** in Eclipse IDE:

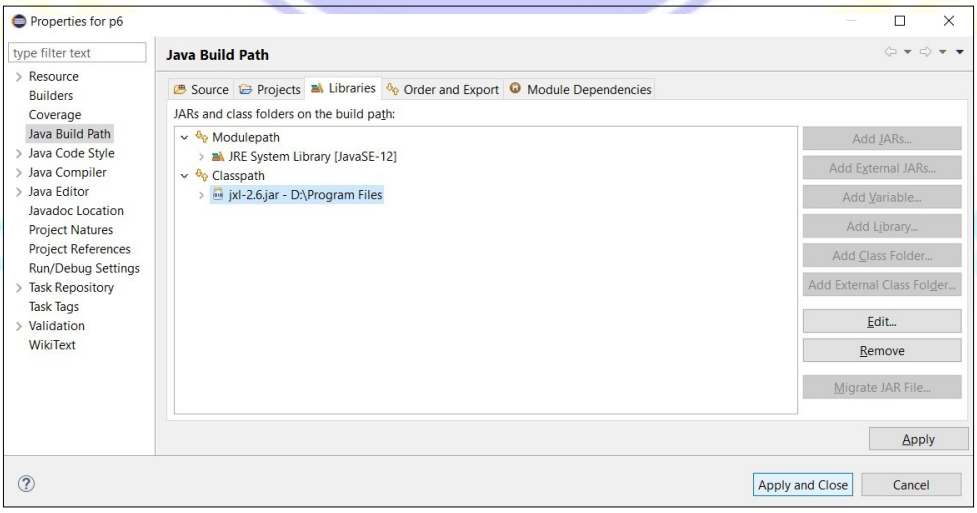
- **right-click on Project Name > Build Path > Configure Build Path…**



- now go under: **Java Build Path > Libraries > Classpath >** click **Add External JARs…**



- Browse and **add JAR file >** click **Apply and Close :**



8) Creating the ***script*** *in JAVA*:

(NOTE that this **script** will be run by Eclipse IDE)

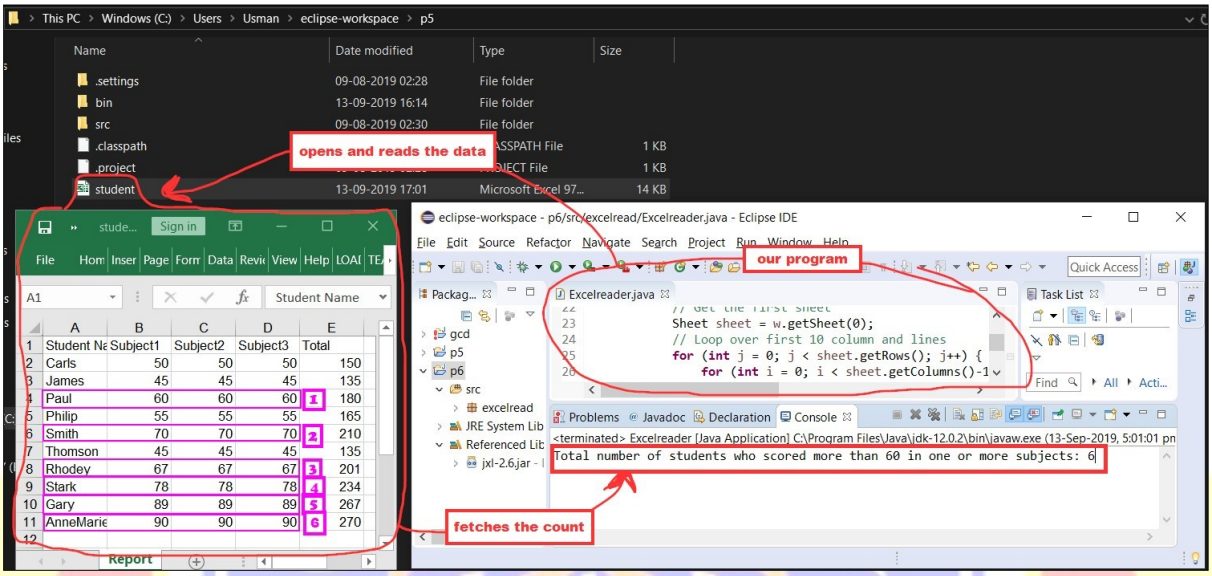
(In simple words, it's like we are
-ordering Eclipse to run a script or to do a job
-of opening .xls file
-and fetching the marks count>=60 from the cells with the help of jxl.jar
-and to show the result.
-Hence automating the work in a local system(PC)).

---(Excelreader.java)---

```
package excelread;


import
java.io.File;
import
java.io.IOExc
eption;import
jxl.Cell;

import
jxl.Ce
llType
;
import
jxl.Sh
eet;
import
jxl.Wo
rkbook
;

import jxl.read.biff.BiffException;


public class Excelreader {

        private String inputFile;

        public void setInputFile(String inputFile) {this.inputFile = inputFile;}


        public void read() throws IOException {

                File inputWorkbook = new
                File(inputFile);Workbook w;

                boolean flag=false;

                int count=0;

                try {

                        w = Workbook.getWorkbook(inputWorkbook);

                        // Get the
                        first sheet
                        Sheet sheet =
                        w.getSheet(0);

                        // Loop over first 10 column and lines
```

```
                    for (int j = 0; j < sheet.getRows(); j++) {
                        for (int i = 0; i <
                            sheet.getColumns()-1; i++)
                            {Cell cell =
                            sheet.getCell(i, j);
                            if (cell.getType() == CellType.NUMBER) {

        if(Integer.parseInt(cell.getContents())>=60){
                                flag = true;
                                if (flag == true){
                                    count++;
                                    flag=false;
                                }
                                Break;
                            }
                        }
                    }
                }
                System.out.println("Total number of students
who scored morethan 60 in one or more subjects: " +count);
            }
            catch (BiffException e) {e.printStackTrace();}
        }
        public static void main(String[] args) throws
            IOException {Excelreader test = new
            Excelreader();
            test.setInputFile("C:\\Users\\Usman\\e
            clipse-
workspace\\p5\\student.xls");
            test.read();
        }
}
```

9) **Run** the file from Eclipse IDE:
   - **OUTPUT:**


opens and reads the data
our program
fetches the count
Total number of students who scored more than 60 in one or more subjects: 6

Finish!

Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) How can Selenium WebDriver be used for extracting data from a web table or form?

2) How do you use Selenium WebDriver to filter students based on their scores in the program?

3) What libraries or methods can you use to perform data analysis in Selenium WebDriver scripts?

4) What steps would you follow to identify students scoring above 60 and display their names in the console?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 6: Selenium Web Driver

Selenium WebDriver is a powerful tool for automating web browser interactions. It allows you to control a browser programmatically, simulating user actions such as clicking, typing, and navigating through web pages. Here's an overview of Selenium WebDriver, including its setup and basic use cases:

1. What is Selenium WebDriver?

- Selenium WebDriver is a part of the Selenium suite, which provides tools for web browser automation.
- It allows for cross-browser testing, meaning you can run the same tests on different browsers like Chrome, Firefox, Safari, and Edge.
- WebDriver interacts directly with the browser's native functionality, providing a more accurate simulation of user actions than older tools like Selenium RC.

2. How Does Selenium WebDriver Work?

- Selenium WebDriver uses browser-specific drivers to interact with the browsers. For example, ChromeDriver is used for Chrome, and GeckoDriver is used for Firefox.
- It communicates with the browser through a web browser driver, which translates the commands in your test script into actions on the browser.

3.**Advantages of Selenium WebDriver**

- Cross-Browser Compatibility: Can run tests on different browsers.
- Support for Multiple Programming Languages: Allows test scripts to be written in Python, Java, C#, Ruby, etc.
- Direct Browser Interaction: Communicates with the browser directly, providing a more accurate automation experience.
- Flexibility in Locating Elements: Offers different methods to locate and interact with elements on a webpage.

4. **Limitations of Selenium WebDriver**

- Not Suitable for Non-Web-Based Applications: Selenium only automates web browsers.
- Handling Complex Actions: Tasks like captcha handling and visual validation are challenging to automate with Selenium alone.
- Browser Dependencies: Requires keeping WebDriver versions in sync with browser versions.

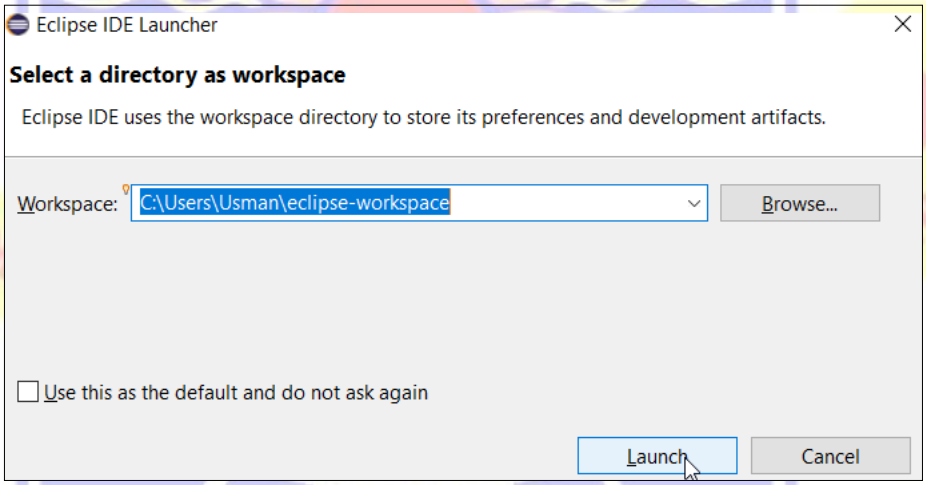
Practical 6: Write a program using Selenium WebDriver to provide the total number of objects present or available on a web page. Perform object identification and counting.

PRE-REQUISITES:

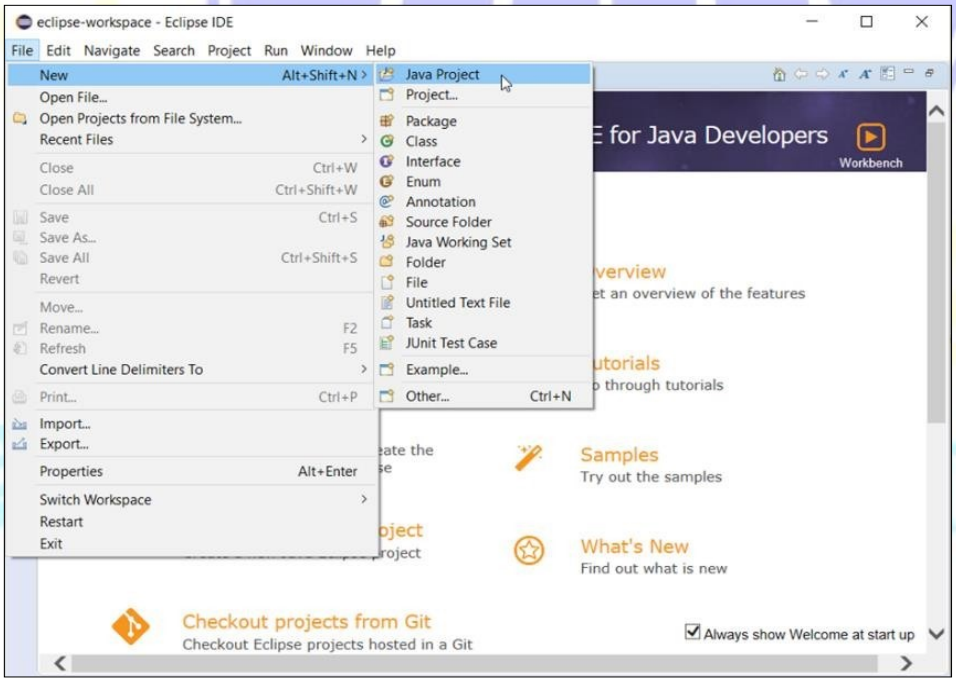
1) Check that you have **JDK**.
2) Check that you have **Eclipse IDE**.
3) Check that you have **Selenium *Server Driver* and *Client Driver*(JAR files)**.
4) Check that you have **Gecko Driver**.
5) Check that you have a stable Internet connection.

STEPS:

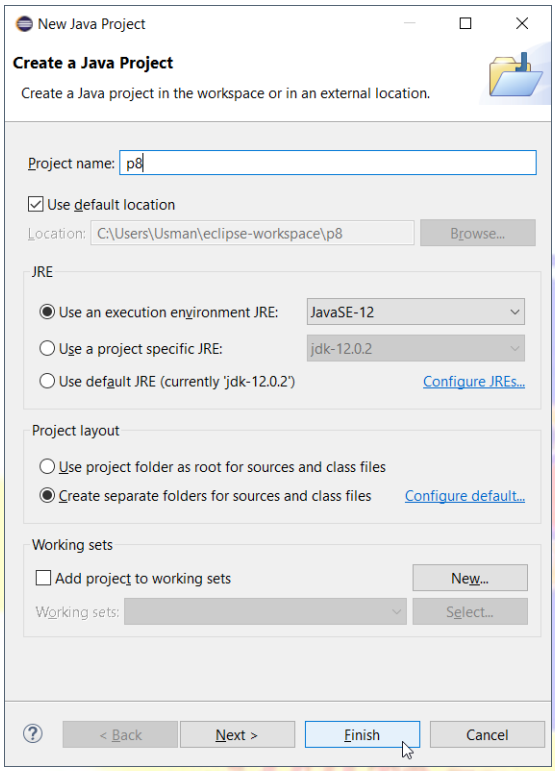
1) **Open Eclipse**. Select your workspace directory. Click **Launch:**

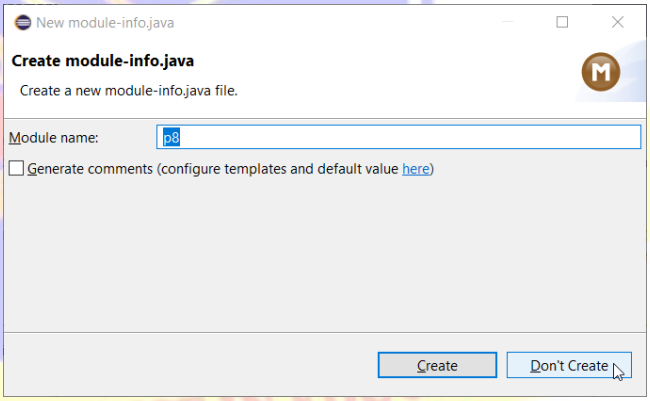


2) **Create** a Project(**File > New > Java Project**):

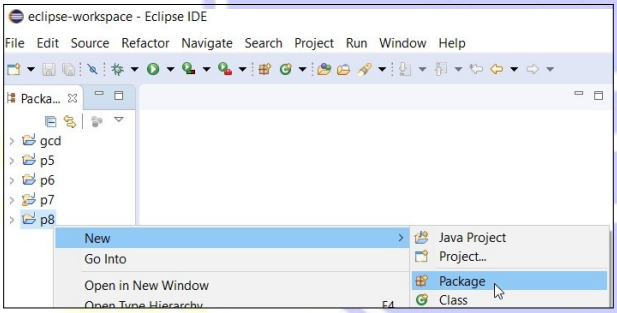


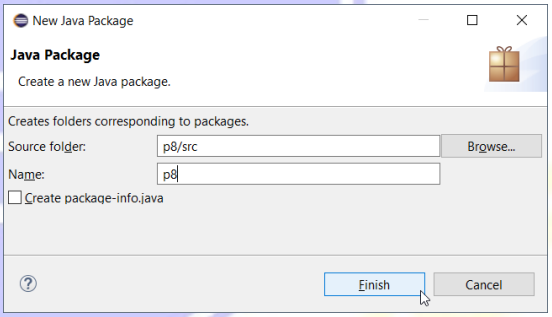
3) **Name the project** as "p8" **>** click **Finish >** click **Don't Create** module:



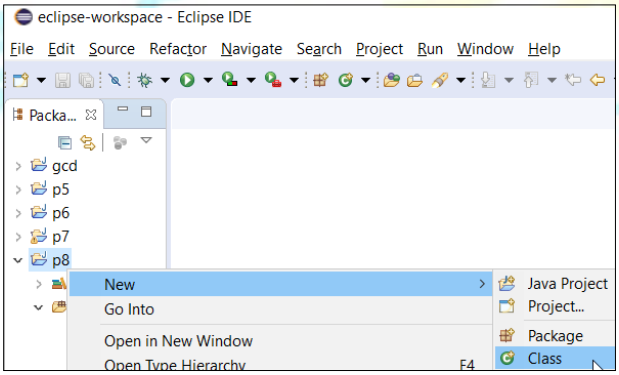


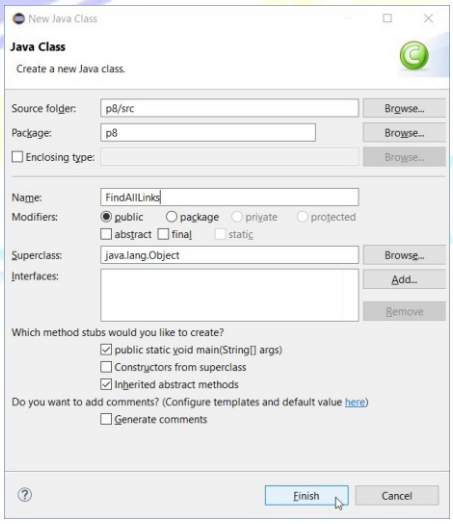
4) **Close the "Welcome" tab.**
5) **Create** a Package(**right-click on Project Name > New > Package > Name** it **> Finish**):





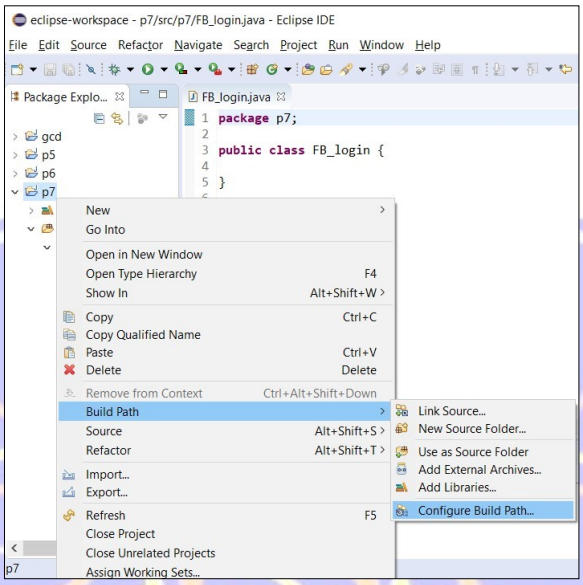
6) **Create** a Class(**right-click on Project Name > New > Class > Name** it **> Finish**):



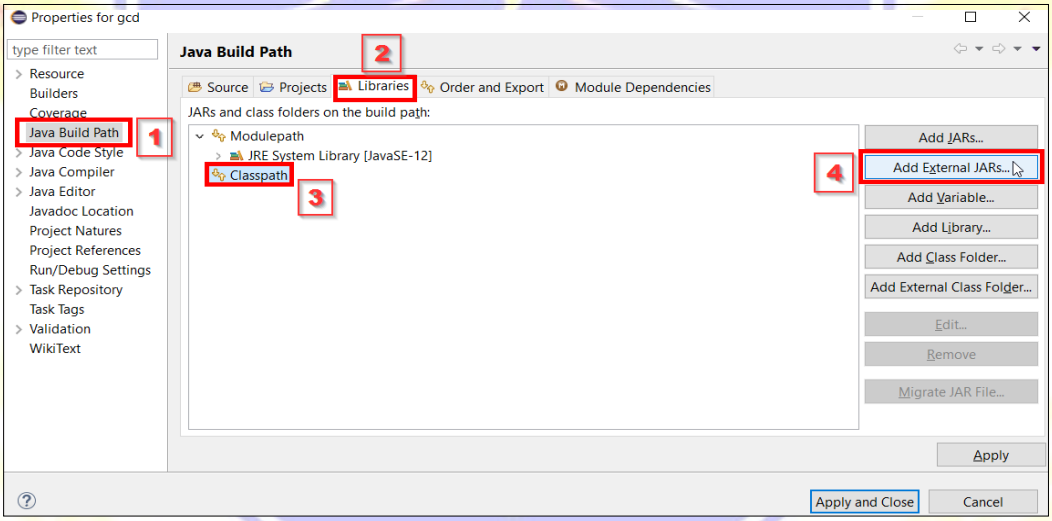


7) Adding **"Selenium *Server Driver* and *Client Driver*(JAR files)"** in Eclipse IDE**:**

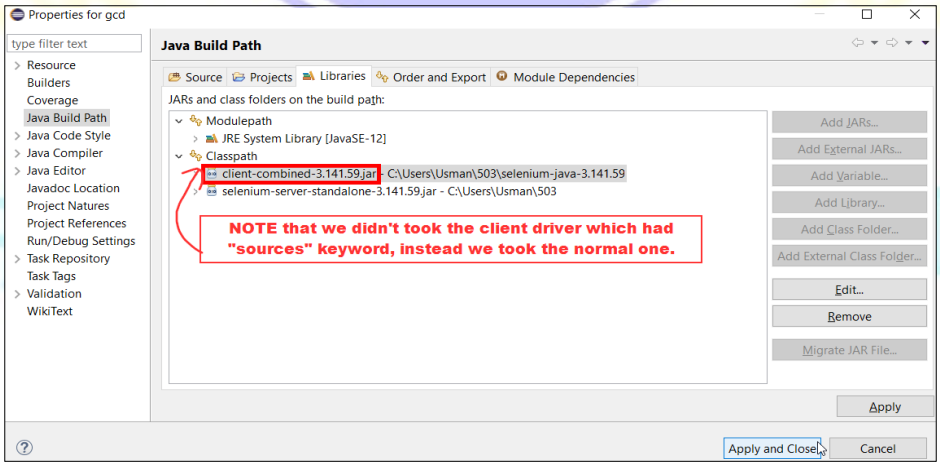
- **right-click on Project Name > Build Path > Configure Build Path…**



- now go under: **Java Build Path > Libraries > Classpath >** click **Add External JARs…**



- Browse and **add JAR files >** click **Apply and Close :**



8) Creating the ***script*** *in JAVA*:

(NOTE that this **script** will be run by Eclipse IDE)
(In simple words, it's like we are
-ordering Eclipse to run a script or to do a job
-of opening the browser, visiting the URL
-and finding the <a> tag WebElements with the help of Selenium Drivers
-and to show the result.
-Hence automating the work in browser)

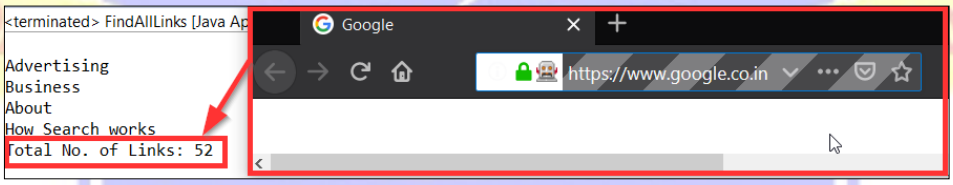
- Now we'll put the path of "geckodriver" in a String ***driverPath***

---(FindAllLinks.java)---

```
package p8;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.firefox.*;

import
org.openqa.selenium.firefox.FirefoxOptions
; import
org.openqa.selenium.firefox.FirefoxProfile
; import
org.openqa.selenium.firefox.internal.Profi
lesIni;

public class FindAllLinks {

	static String driverPath = "C:\\Users\\Usman\\503\\geckodriver.exe";

	public static void main(String[] args) {
		System.setProperty("webdriver.gecko.driver",driverPath);

		//NOTE THAT: following commented lines are required for old machines

		//DesiredCapabilities capabilities = DesiredCapabilities.firefox();

		//capabilities.setCapability("marionette",true);

		//ProfilesIni allProfiles = new ProfilesIni();

		//FirefoxProfile fp = new FirefoxProfile();

		//fp.setPreference(FirefoxProfile.PORT_PREFERENCE,"7055");

		//FirefoxOptions options = new FirefoxOptions();

		//options.setProfile(fp);

		WebDriver driver = new
		FirefoxDriver(); String
		appUrl
		="https://www.google.co.in/";
		driver.get(appUrl);
```

```
            java.util.List<WebEl
ement> links =
driver.findElements(By.tagName("a
"));

            for (int i = 1; i<links.size(); i=i+1)
            {
                System.out.println(links.get(i).getText());
            }
            System.out.println("Total No. of Links: "+links.size());
            //driver.quit();
        }
}
```

9) Run the file from Eclipse IDE: OUTPUT:



Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) How can Selenium WebDriver be used to identify all objects or elements on a web page?

2) What method would you use to count the total number of objects found on a web page using Selenium?

3) How do you differentiate between different types of web elements while counting them using Selenium WebDriver?

4) What challenges might you face when counting elements on a dynamic web page, and how can they be handled?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 7: Selenium Web Driver

Selenium WebDriver is a powerful tool for automating web browser interactions. It allows you to control a browser programmatically, simulating user actions such as clicking, typing, and navigating through web pages. Here's an overview of Selenium WebDriver, including its setup and basic use cases:

1. What is Selenium WebDriver?

- Selenium WebDriver is a part of the Selenium suite, which provides tools for web browser automation.
- It allows for cross-browser testing, meaning you can run the same tests on different browsers like Chrome, Firefox, Safari, and Edge.
- WebDriver interacts directly with the browser's native functionality, providing a more accurate simulation of user actions than older tools like Selenium RC.

2. How Does Selenium WebDriver Work?

- Selenium WebDriver uses browser-specific drivers to interact with the browsers. For example, ChromeDriver is used for Chrome, and GeckoDriver is used for Firefox.
- It communicates with the browser through a web browser driver, which translates the commands in your test script into actions on the browser.

3.**Advantages of Selenium WebDriver**

- Cross-Browser Compatibility: Can run tests on different browsers.
- Support for Multiple Programming Languages: Allows test scripts to be written in Python, Java, C#, Ruby, etc.
- Direct Browser Interaction: Communicates with the browser directly, providing a more accurate automation experience.
- Flexibility in Locating Elements: Offers different methods to locate and interact with elements on a webpage.

4. **Limitations of Selenium WebDriver**

- Not Suitable for Non-Web-Based Applications: Selenium only automates web browsers.
- Handling Complex Actions: Tasks like captcha handling and visual validation are challenging to automate with Selenium alone.
- Browser Dependencies: Requires keeping WebDriver versions in sync with browser versions.

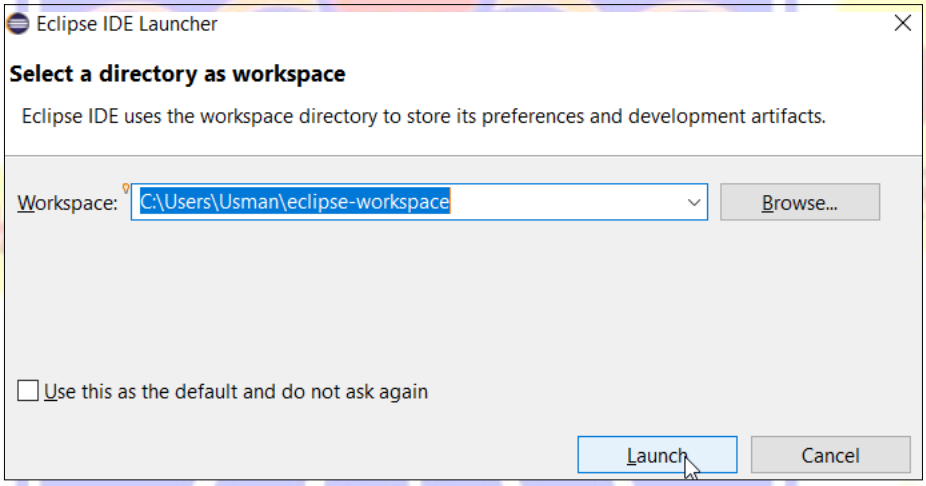
Practical 7: Write a program using Selenium WebDriver to get the number of items in a list or combo box on a web page. Perform element identification and counting

PRE-REQUISITES:

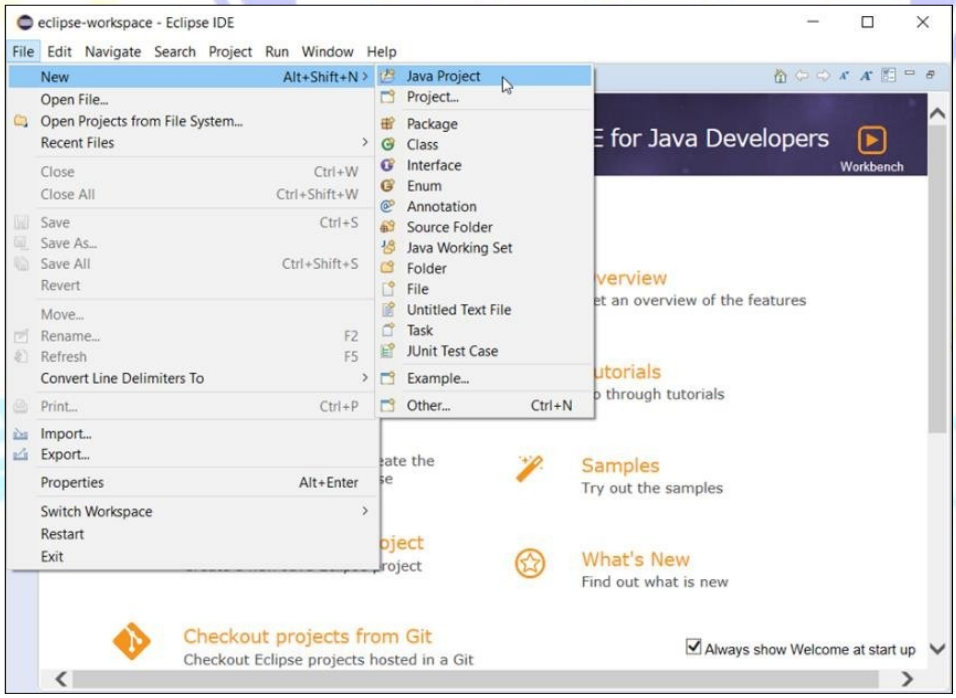
1) Check that you have **JDK**.
2) Check that you have **Eclipse IDE**.
3) Check that you have **Selenium *Server Driver* and *Client Driver*(JAR files)**.
4) Check that you have **Gecko Driver**.
5) Check that you have a stable Internet connection.

STEPS:

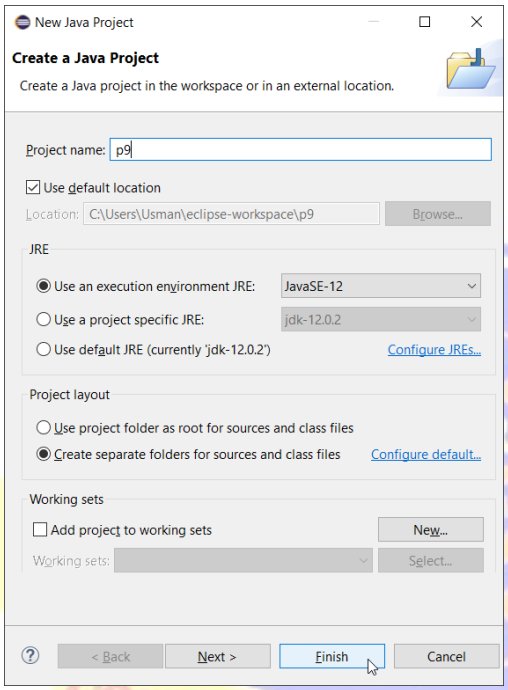
1) **Open Eclipse**. Select your workspace directory. Click **Launch:**

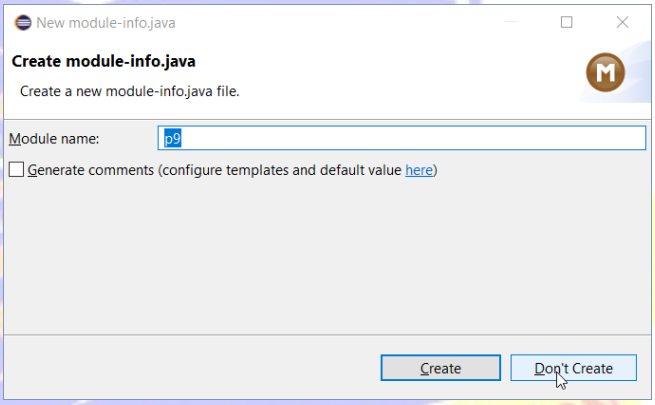


2) **Create** a Project(**File > New > Java Project**):



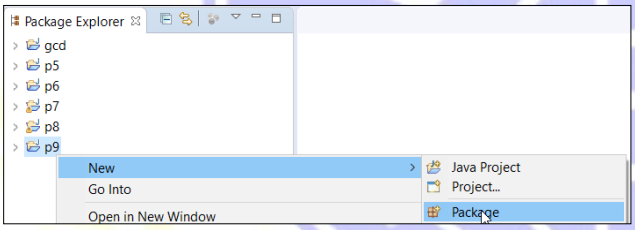
3) **Name the project** as "p9" **>** click **Finish >** click **Don't Create** module:

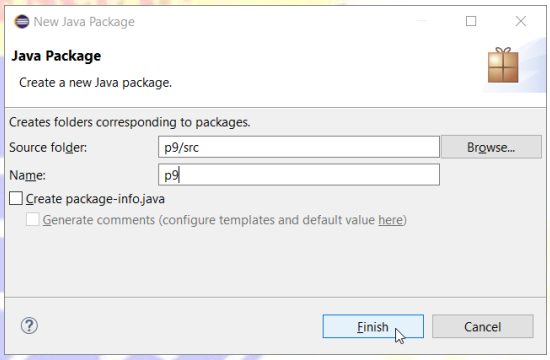




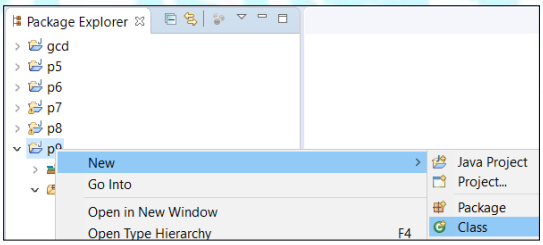
4) **Close the "Welcome" tab.**

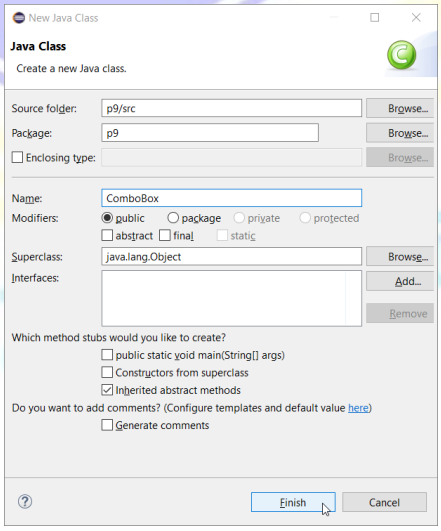
5) **Create** a Package(**right-click on Project Name > New > Package > Name** it **> Finish**):





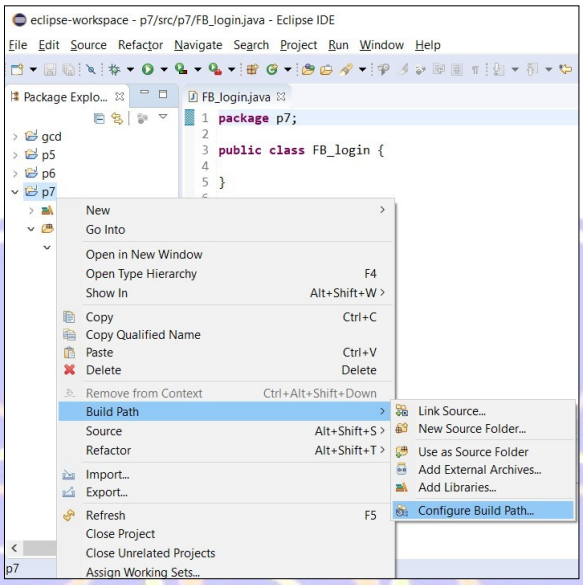
6) **Create** a Class(**right-click on Project Name > New > Class > Name** it **> Finish**):



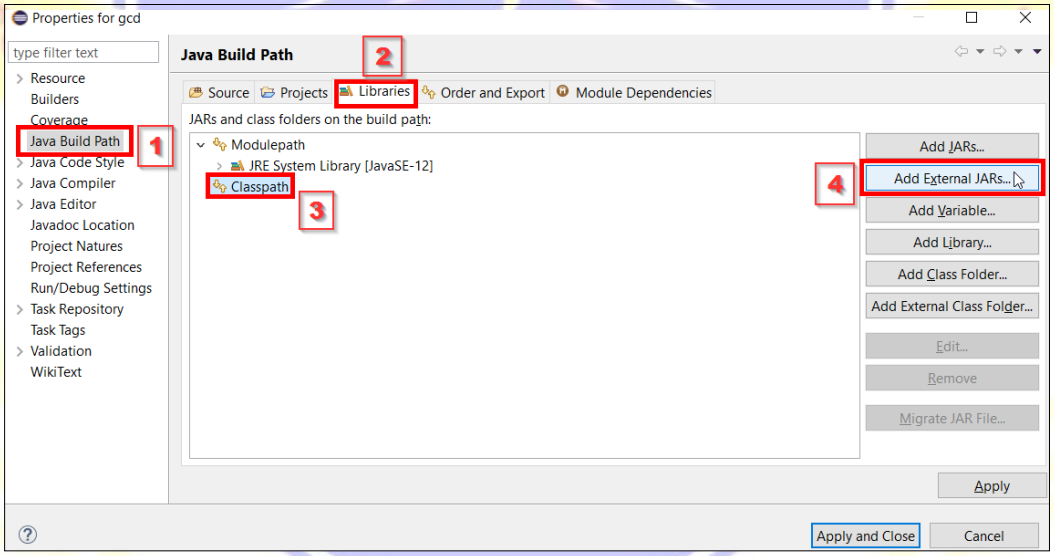


7) Adding **"Selenium *Server Driver* and *Client Driver*(JAR files)"** in Eclipse IDE**:**

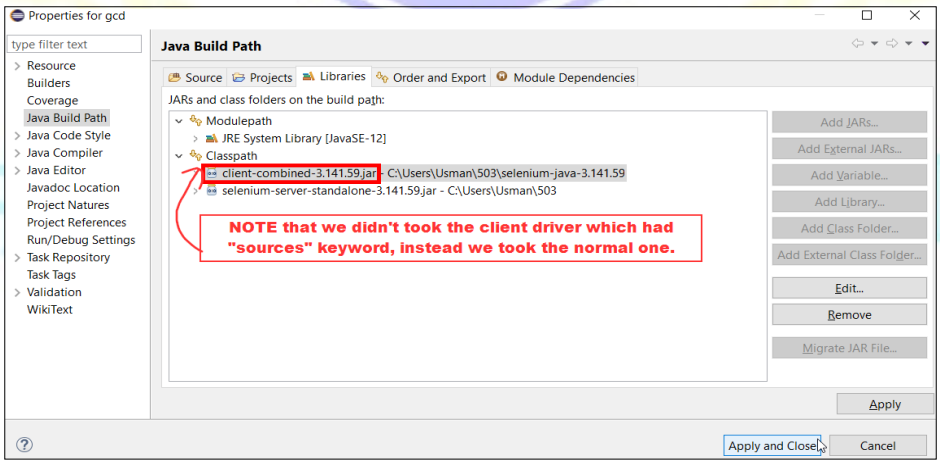
- **right-click on Project Name > Build Path > Configure Build Path…**



- now go under: **Java Build Path > Libraries > Classpath >** click **Add External JARs…**



- Browse and **add JAR files >** click **Apply and Close :**

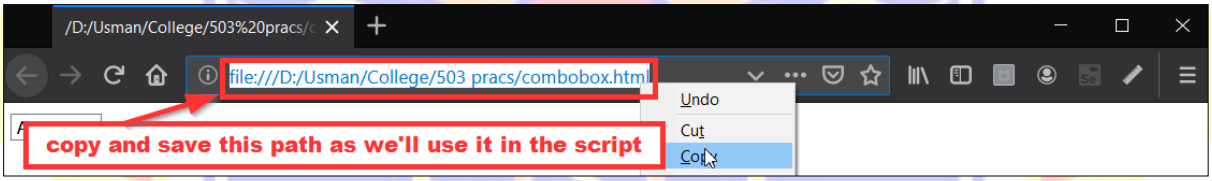


Create HTML file in a notepad **> Save it > Open it in browser > Copy the URL:**

(NOTE THAT: as this is our LOCAL FILE, this file will be opened thru STATIC URL in script)

---(combobox.html)---

```
<select id="continents">
        <option value="Asia">Asia</option>
        <option value="Europe">Europe</option>
        <option value="Africa">Africa</option>
</select>
```



8) Creating the ***script*** *in JAVA*:
(NOTE that this **script** will be run by Eclipse IDE)
(In simple words, it's like we are
-ordering Eclipse to run a script or to do a job
-of opening the browser, visiting the URL
-and finding the WebElement By "ID" with the help of "Select" class and Selenium Drivers
-and to show the result.
-Hence automating the work in browser)

- Now we'll put the path of our LOCAL FILE(combobox.html) in a string
- Also put the path of "geckodriver" in a String ***driverPath***

---(ComboBox.java)---

```
package p9;

import java.util.List;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.firefox.*;
import
org.openqa.selenium.firefox.FirefoxOptions
; import
org.openqa.selenium.firefox.FirefoxProfile
; import
org.openqa.selenium.firefox.internal.Profi
lesIni;import
org.openqa.selenium.support.ui.Select;
```

```
public class ComboBox {

        static String driverPath = "C:\\Users\\Usman\\503\\geckodriver.exe";

        public static void main(String[] args) {
            System.setProperty("webdriver.gecko.driver",driverPath);

            //NOTE THAT: following commented lines are required for old machines

            //DesiredCapabilities capabilities = DesiredCapabilities.firefox();

               //capabilities.setCapability("marionette",true);

               //ProfilesIni allProfiles = new ProfilesIni();

               //FirefoxProfile fp = new FirefoxProfile();

               //fp.setPreference(FirefoxProfile.PORT_PREFERENCE,"7055");

               //FirefoxOptions options = new FirefoxOptions();

               //options.setProfile(fp);

               WebDriver driver = new FirefoxDriver();

               String appUrl ="https://www.toolsqa.com/automation-practice-form/";
//DYNAMIC URL(WEBSITE)

               //String appUrl = "
file:///D:/Usman/College/503%20pracs/combobox.html"; //STATIC
URL(LOCAL FILE)

               driver.get(appUrl);

               Select oSelect = new

Select(driver.findElement(By.id("continents")));

               //Select oSelect = new
Select(driver.findElement(By.tagName("select"))); //this works
too

               List<WebElement> oSize = oSelect.getOptions();

               int iListSize = oSize.size();

               for(int i =0; i < iListSize ; i++)

               {

                     // Storing the value of the option

                     String sValue = oSelect.getOptions().get(i).getText();

                     // Printing
                     the stored
                     value
                     System.out.pri
                     ntln(sValue);

               }
```

```
            System.out.println("Total No. Items in Dropdown: "+iListSize);
            //driver.quit();
        }
}
```

9) **Run the file from Eclipse IDE:**

- **OUTPUT:**

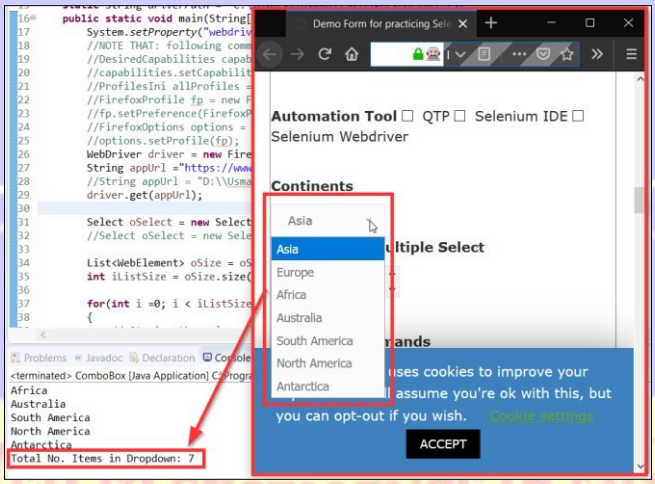
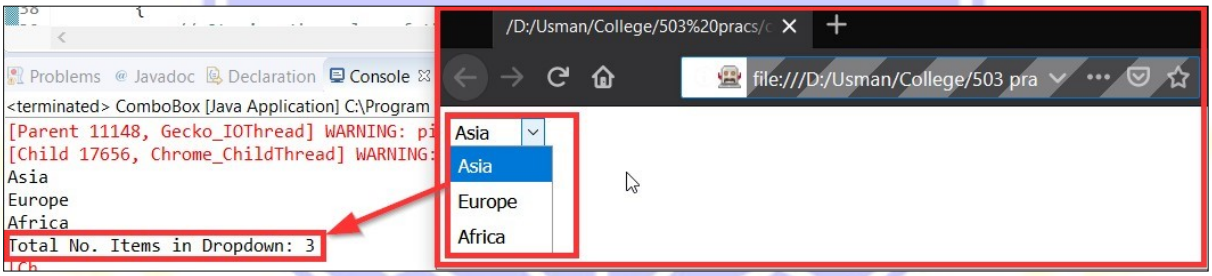


Figure 1: by using the DYNAMIC URL(website)



Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) How can you locate a combo box or a list element on a web page using Selenium WebDriver?
2) What method would you use to retrieve all the options available in a combo box or list using Selenium WebDriver?

3) How do you count the total number of items in a combo box or list using Selenium WebDriver?

4) What challenges might arise when working with dynamic combo boxes or lists, and how can they be addressed?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]

Theory 8: Selenium Web Driver

Selenium WebDriver is a powerful tool for automating web browser interactions. It allows you to control a browser programmatically, simulating user actions such as clicking, typing, and navigating through web pages. Here's an overview of Selenium WebDriver, including its setup and basic use cases:

1. What is Selenium WebDriver?

- Selenium WebDriver is a part of the Selenium suite, which provides tools for web browser automation.
- It allows for cross-browser testing, meaning you can run the same tests on different browsers like Chrome, Firefox, Safari, and Edge.
- WebDriver interacts directly with the browser's native functionality, providing a more accurate simulation of user actions than older tools like Selenium RC.

2. How Does Selenium WebDriver Work?

- Selenium WebDriver uses browser-specific drivers to interact with the browsers. For example, ChromeDriver is used for Chrome, and GeckoDriver is used for Firefox.
- It communicates with the browser through a web browser driver, which translates the commands in your test script into actions on the browser.

3.**Advantages of Selenium WebDriver**

- Cross-Browser Compatibility: Can run tests on different browsers.
- Support for Multiple Programming Languages: Allows test scripts to be written in Python, Java, C#, Ruby, etc.
- Direct Browser Interaction: Communicates with the browser directly, providing a more accurate automation experience.
- Flexibility in Locating Elements: Offers different methods to locate and interact with elements on a webpage.

4. **Limitations of Selenium WebDriver**

- Not Suitable for Non-Web-Based Applications: Selenium only automates web browsers.
- Handling Complex Actions: Tasks like captcha handling and visual validation are challenging to automate with Selenium alone.
- Browser Dependencies: Requires keeping WebDriver versions in sync with browser versions.

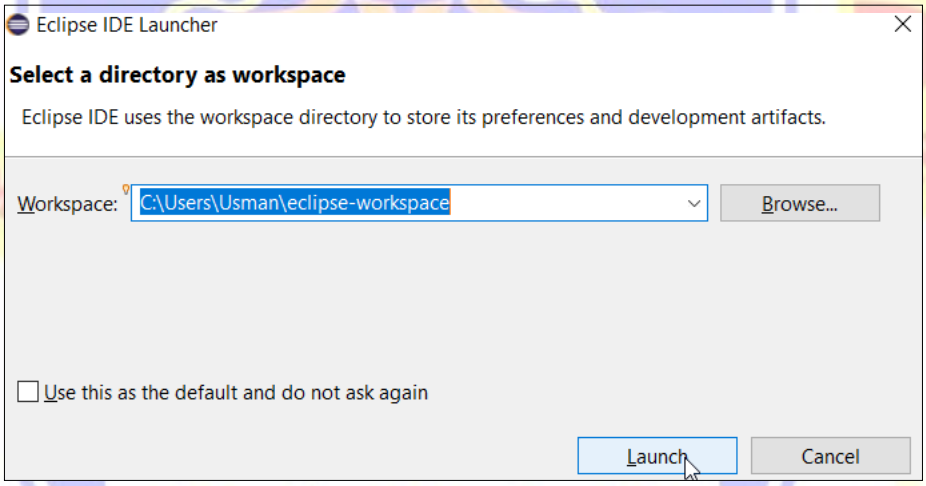
Practical 8: Write a program using Selenium WebDriver to count the number of checkboxes on a web page, including checked and unchecked counts. Perform checkbox identification and counting.

PRE-REQUISITES:

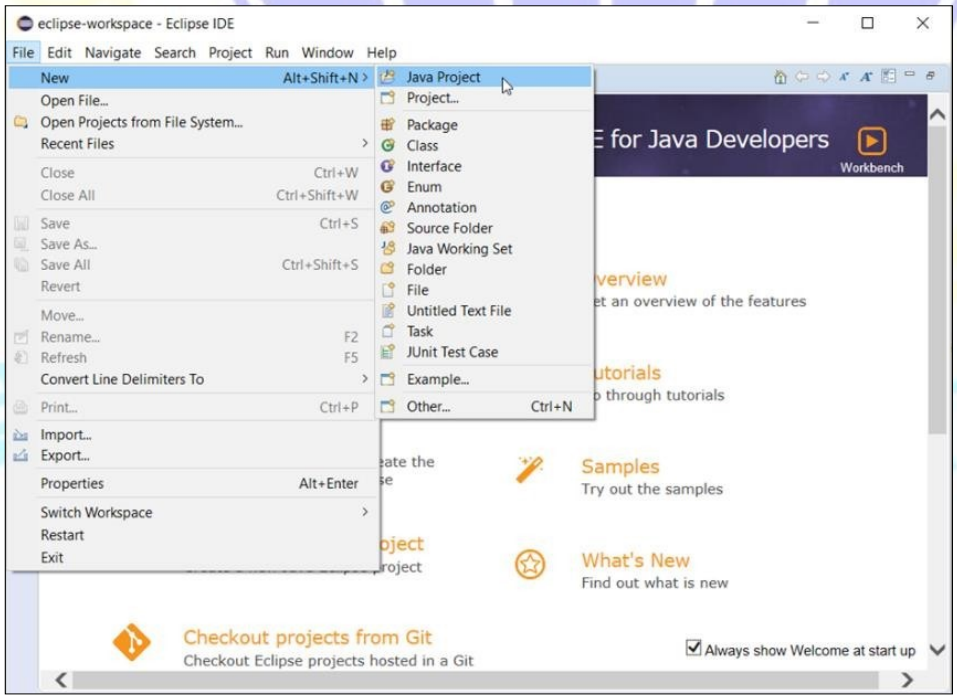
1) Check that you have **JDK**.
2) Check that you have **Eclipse IDE**.
3) Check that you have **Selenium *Server Driver* and *Client Driver*(JAR files)**.
4) Check that you have **Gecko Driver**.
5) Check that you have a stable Internet connection.

STEPS:

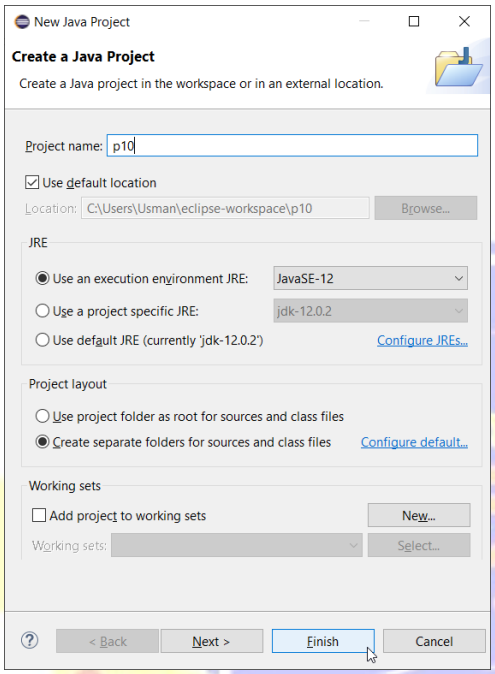
1) **Open Eclipse**. Select your workspace directory. Click **Launch:**

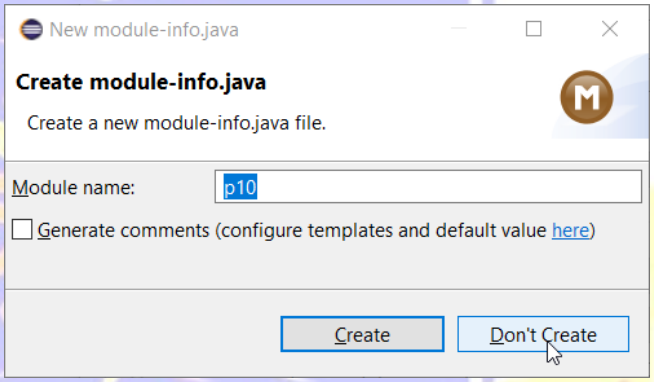


2) **Create** a Project(**File > New > Java Project**):



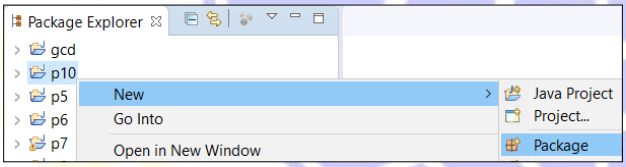
3) **Name the project** as "p10" **>** click **Finish >** click **Don't Create** module:

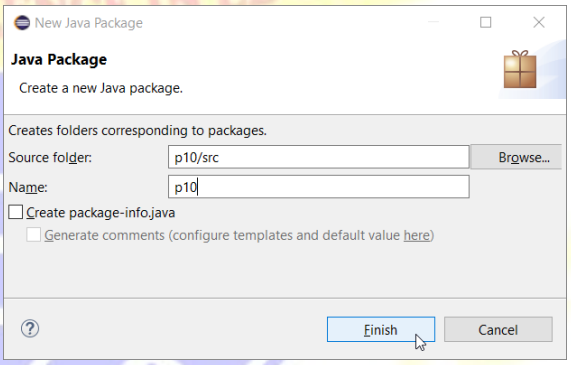




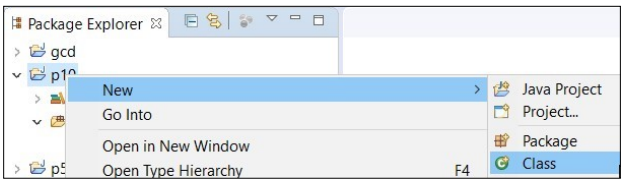
4) **Close the "Welcome" tab.**

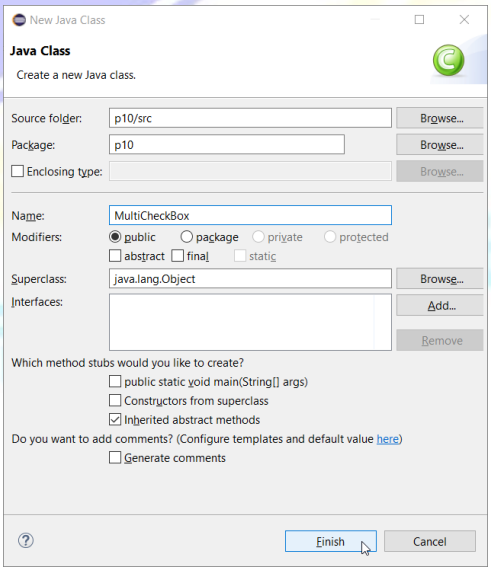
5) **Create** a Package(**right-click on Project Name > New > Package > Name** it **> Finish**):





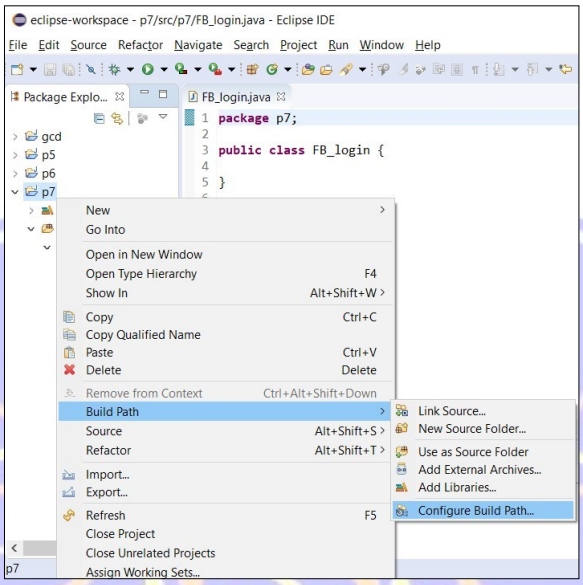
6) **Create** a Class(**right-click on Project Name > New > Class > Name** it **> Finish**):



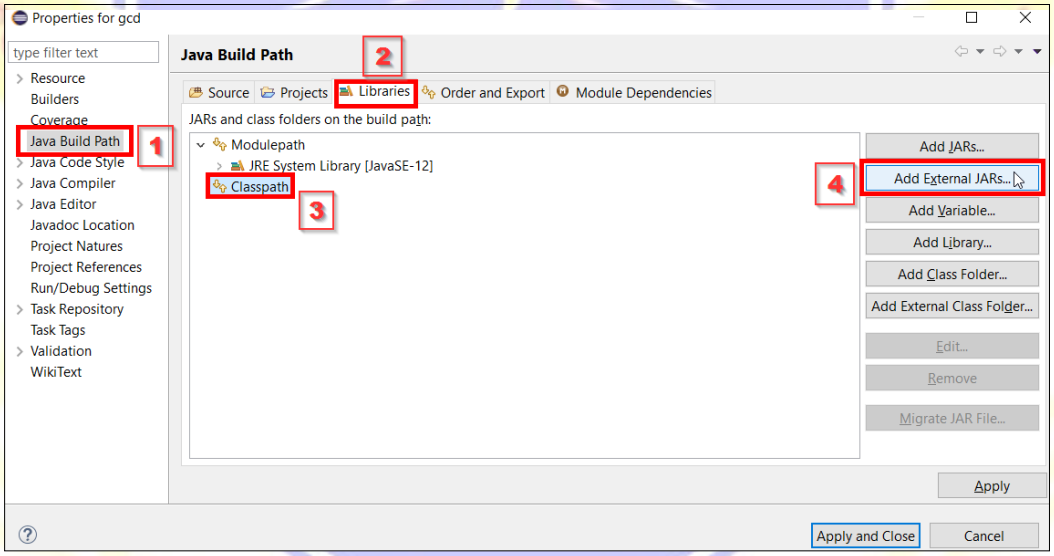


7) Adding **"Selenium *Server Driver* and *Client Driver*(JAR files)"** in Eclipse IDE**:**

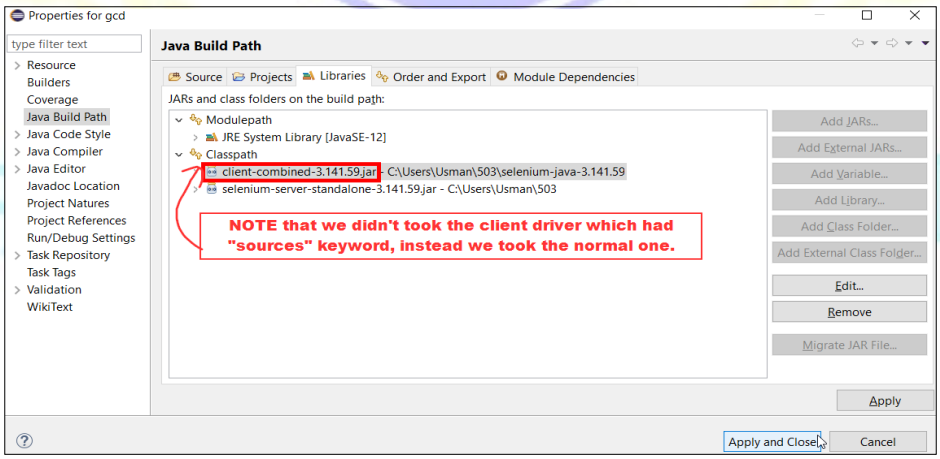
- **right-click on Project Name > Build Path > Configure Build Path…**



- now go under: **Java Build Path > Libraries > Classpath >** click **Add External JARs…**



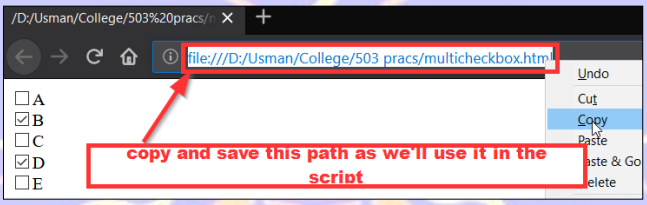
- Browse and **add JAR files >** click **Apply and Close :**



8) **Create** HTML file in a notepad **> Save it > Open it in browser > Copy the URL:**
(NOTE THAT: as this is our LOCAL FILE, this file will be opened thru STATIC URL in script)

---(multicheckbox.html)---

```
<input type="checkbox" value="A">A<br>

<input type="checkbox" value="B" CHECKED>B<br>

<input type="checkbox" value="C">C<br>

<input type="checkbox" value="D" CHECKED>D<br>

<input type="checkbox" value="E">E<br>
```



9) Creating the ***script*** *in JAVA*:
(NOTE that this **script** will be run by Eclipse IDE)
(In simple words, it's like we are
-ordering Eclipse to run a script or to do a job
-of opening the browser, visiting the URL
-and finding the WebElement By "xpath" with the help of Selenium Drivers
-and to show the result.
-Hence automating the work in browser)

- Now we'll put the path of our LOCAL FILE(multicheckbox.html) in a string
- Also put the path of "geckodriver" in a String ***driverPath***

---(MultiCheckBox.java)---

```
package p10;

import java.util.List;

import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.firefox.*;

import
org.openqa.selenium.firefox.FirefoxOptions
; import
org.openqa.selenium.firefox.FirefoxProfile
; import
org.openqa.selenium.firefox.internal.Profi
lesIni;

public class MultiCheckBox {

        static String driverPath = "C:\\Users\\Usman\\503\\geckodriver.exe";
```

```
        public static void main(String[] args) {
                System.setProperty("webdriver.gecko.driver",driverPath);

                //DesiredCapabilities capabilities = DesiredCapabilities.firefox();

                //capabilities.setCapability("marionette",true);

                //ProfilesIni allProfiles = new ProfilesIni();

                //FirefoxProfile fp = new FirefoxProfile();

                //fp.setPreference(FirefoxProfile.PORT_PREFERENCE,"7055");

                //FirefoxOptions options = new FirefoxOptions();

                //options.setProfile(fp);

                WebDriver driver = new FirefoxDriver();

        String appUrl = "https://www.toolsqa.com/automation-practice-form/";

//DYNAMIC URL(WEBSITE)

                //String appUrl =
"file:///D:/Usman/College/503%20pracs/multicheckbox.html";
//STATIC URL(LOCALFILE)

                driver.get(appUrl);


                List<WebElement> checkBoxes =
driver.findElements(By.xpath("//input[@type='checkbox']"));


                int checkedCount=0, uncheckedCount=0;

                for(int i=0; i<checkBoxes.size(); i++){

                        System.out.println(i+"
checkbox is selected
"+checkBoxes.get(i).isSelected());

                        if(checkBoxes.get(i).isSelected()) {checkedCount++;}

                        else {uncheckedCount++;}

                }

                //driver.quit();

                System.out.println("No. of selected checkbox: "+checkedCount);
                System.out.println("No. of unselected checkbox: "+uncheckedCount);

        }

}
```

10) Run the file from Eclipse IDE:

- **OUTPUT:**

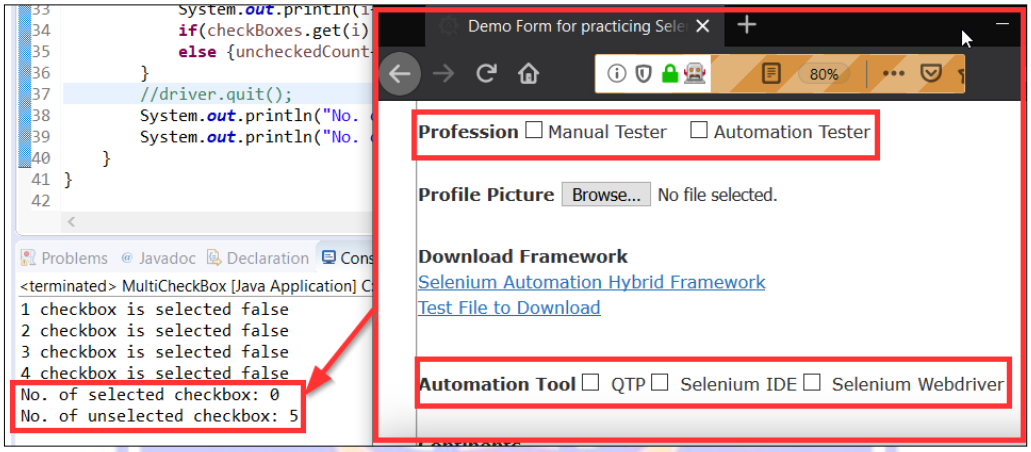


Figure 1: by using the DYNAMIC URL(website)

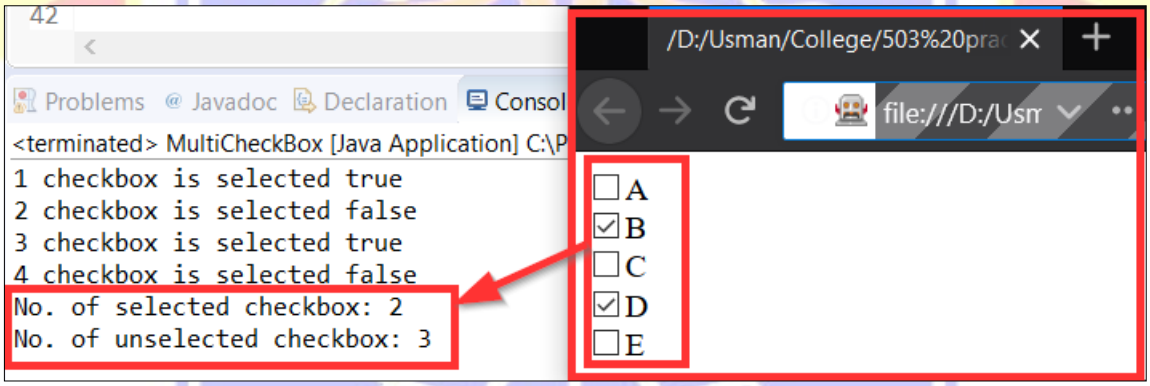


Figure 2: by using the STATIC URL(local file)

Result and Discussion:

Learning Outcomes:

Course Outcome:

Conclusion:

Viva Questions:

1) How can you identify checkboxes on a web page using Selenium WebDriver?

2) What method would you use to check whether a checkbox is selected or not in Selenium WebDriver?

3) How can you count the total number of checkboxes and the number of checked and unchecked ones?

4) What are the challenges when dealing with dynamically loaded checkboxes, and how can they be handled in Selenium WebDriver?

For Faculty Use

Correction Parameters	Formative Assessments [ ]	Timely completion of Practical [ ]	Attendance Learning Attitude [ ]