

SVERI's College of Engineering, Pandharpur

Department of Computer Science and Engineering



Lab Manual of SOFTWARE TESTING & QUALITY ASSURANCE (STQA) LY B.Tech (CSE) Sem-I

Prepared by:

Prof. A. K. MORE

2023-24

VISION	
Institute Vision	Department Vision
To be nationally recognized among the best institutes in India for excellence in technical education	To be nationally recognized for excellence in education augmented by research in the field of Computer Science and Engineering
MISSION	
Institute Mission	Department Mission
<ul style="list-style-type: none"> To impart value-based technical education through innovation and excellence, empowering individuals to become leaders in their fields to create positive impact. To create an ambiance of academic excellence, research, and life skills by fostering a learning environment that empowers individuals to achieve their full potential. To foster strong relationships amongst all our stakeholders by inculcating a personal touch and mutual respect in all our interactions. 	<p>To impart value-based education in Computer Science and Engineering, through effective teaching and learning approaches.</p> <p>To create ambiance for academic excellence through fruitful interaction among various stakeholders.</p> <p>To inculcate best practices for innovative research, competitive employability and sustainable entrepreneurship development.</p>
PROGRAMME EDUCATIONAL OBJECTIVES (PEOs)	
<p>The Department of Computer Science and Engineering has as its PEOs to produce graduate who:</p> <ol style="list-style-type: none"> 1. Apply the Computer Science domain specific knowledge and skills in the growing software and related industries. 2. Demonstrate leadership, professional ethics, project management and finance related attributes as employees or employers. 3. Engage in life-long learning for professional advancement to develop innovative solutions for individual or societal problems. 4. Demonstrate strong communication skills and ability to function effectively as an individual and part of a team. 	

PROGRAMME SPECIFIC OUTCOMES (PSOs)

Engineering Graduates will be able to:

1. Understand & design computer system using knowledge of Digital Techniques, Micro-Processor, Computer Organization, Advanced Computer Architecture, Operating System, System Programming, Compiler Construction, Application Softwares, etc.
2. Interpret, analyze and design software system programming knowledge using Algorithmic Skills, Web Technology, Big Data Analytics, Networking Fundamentals, Machine Learning and Internet of Things.
3. Adopt applications in emerging fields of Computer Science & Engineering.

PROGRAMME OUTCOMES (POs)	
Students graduating from Computer Science and Engineering will demonstrate:	
1	Engineering knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2	Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3	Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4	Conduct investigations of complex problems: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5	Modern tool usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.
6	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7	The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
8	Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9	Individual and team work: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10	Communication: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11	Project management and finance: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments
12	Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Outcome (CO):

1. Identify what a software bug is, how serious they can be, and why they occur.
2. Test software to meet quality objectives & requirements.
3. Apply testing skills to common testing tasks.
4. Perform the planning and documentation of test efforts.
5. Describe software quality concepts, assurance & standards.
6. Use testing tools to test software in order to improve test efficiency with automation.

Experiment List

Exp t. no.	Experiment Title	CO	BL	PI
1.	Introduction to Manual Testing and Understand differences between Manual testing and Automation Testing.	CS414-21.1	BL 1	1.7.1
2.	Understanding of test case parameters.	CS414-21.2	BL 2	1.7.1
3.	Implementation of test case using manual testing.	CS414-21.4	BL 2	3.5.6
4.	Introduction to Selenium (WebDriver).	CS414-21.3	BL 3	5.4.1
5.	Implements the Steps to install Selenium WebDriver.	CS414-21.3	BL 3	5.6.1
6.	Program to Implement of Launching the browser.	CS414-21.3	BL 3	5.4.2
7.	Understand Different commands in Selenium.(drive.get(),driver.nevigate.to(),close(),quite(), FindElement&s())	CS414-21.3	BL 3	5.5.1
8.	Understanding Different Types of Locators in Selenium.(id,name,xpath)	CS414-21.3	BL 3	5.5.2
9.	Handling Alerts in Selenium WebDriver.	CS414-21.3	BL 3	5.5.2
10.	Mini project	CS414-21.3	BL 3	9.4.2

Experiment No -1

Aim: Explain Manual Testing? Write differences between Manual testing and Automation Testing.

Manual Testing

- Manual testing is a software testing process in which test cases are executed manually without using any automated tool. All test cases executed by the tester manually according to the end user's perspective. It ensures whether the application is working, as mentioned in the requirement document or not. Test cases are planned and implemented to complete almost 100 percent of the software application. Test case reports are also generated manually.
- Manual Testing is one of the most fundamental testing processes as it can find both visible and hidden defects of the software. The difference between expected output and output, given by the software, is defined as a defect. The developer fixed the defects and handed it to the tester for retesting.
- Manual testing is mandatory for every newly developed software before automated testing. This testing requires great efforts and time, but it gives the surety of bug-free software. Manual Testing requires knowledge of manual testing techniques but not of any automated testing tool.
- Manual testing is essential because one of the software testing fundamentals is "100% automation is not possible."
- Types of Manual Testing
 - White Box Testing
 - Black Box Testing
 - Gray Box Testing

Steps to perform Manual Testing:

- First, tester observes all documents related to software, to select testing areas.
- Tester analyses requirement documents to cover all requirements stated by the customer.
- Tester develops the test cases according to the requirement document.
- All test cases are executed manually by using Black box testing and white box testing.
- If bugs occurred, then the testing team informs the development team.
- The Development team fixes bugs and handed software to the testing team for a retest.

Differences between Manual testing and Automation Testing:

Manual testing	Automation Testing
In manual testing, test cases are executed by a human tester and software.	Automation Testing uses automation tools to execute test cases.
Manual testing is time-consuming and takes up human resources.	Automated testing is significantly faster than a manual approach.
Automation does not allow random testing	Exploratory testing is possible in Manual Testing
The initial investment in the automated testing is higher. Though the ROI is better in the long run.	The initial investment in the Manual testing is comparatively lower. ROI is lower compared to Automation testing in the long run.
Automated testing is a reliable method, as it is performed by tools and scripts. There is no testing Fatigue.	Manual testing is not as accurate because of the possibility of the human errors.
For even a trivial change in the UI of the AUT, Automated Test Scripts need to be modified to work as expected	Small changes like change in id, class, etc. of a button wouldn't thwart execution of a manual tester.
Investment is required for testing tools as well as automation engineers	Investment is needed for human resources.
Not cost effective for low volume regression	Not cost effective for high volume regression.
With automation testing, all stakeholders can login into the automation system and check test execution results	Manual Tests are usually recorded in an Excel or Word, and test results are not readily/ readily available.
Automated testing does not involve human consideration. So it can never give assurance of user-friendliness and positive customer experience.	The manual testing method allows human observation, which may be useful to offer user-friendly system.
Performance Tests like Load Testing, Stress Testing, Spike Testing, etc. have to be tested by an automation tool compulsorily.	Performance Testing is not feasible manually

Conclusion: Students are able to understand differences between manual and automation testing.

Experiment No -2

Aim: What is test case. What are the key points to write a good test case?

Test Case

- A test case is a document, which has a set of test data, preconditions, expected results and post conditions, developed for a particular test scenario in order to verify compliance against a specific requirement.
- Test Case acts as the starting point for the test execution, and after applying a set of input values, the application has a definitive outcome and leaves the system at some end point or also known as execution post condition.

Typical Test Case Parameters

- Test Case ID
- Test Scenario
- Test Case Description
- Test Steps
- Prerequisite
- Test Data
- Expected Result
- Test Parameters
- Actual Result
- Environment Information
- Comments

Example:

Let us say that we need to check an input field that can accept maximum of 10 characters.

In the below example, the first case is a pass scenario while the second case is a FAIL.

Scenario	Test Step	Expected Result	Actual Outcome
Verify that the input field that can accept maximum of 10 characters	Login to application and key in 10 characters	Application should be able to accept all 10 characters.	Application accepts all 10 characters.
Verify that the input field that can accept maximum of 11 characters	Login to application and key in 11 characters	Application should NOT accept all 11 characters.	Application accepts all 10 characters.

If the expected result doesn't match with the actual result, then we log a defect. The defect goes through the defect life cycle and the testers address the same after fix.

Key points to write a good Test Case

- Keep things simple and transparent.
- Make test cases reusable.
- Keep test case IDs unique.
- Peer review is important.
- Test cases should have the end user or defined requirements in mind.
- Specify expected results and assumptions.

Conclusion : Students are able to understand test case is and what are the key points to write a good

Experiment No -3

Aim: Write a test case using manual testing.

Example 1: Write a test using manual testing

Prerequisite: None

Outcome: Students should be able to understand manual testing and creation of test cases to test a login window.

Test Cases:

Steps	Expected results
Enter valid username and valid password click login button	The application should display the home page
log out enter valid username and an invalid password click login button	The application should display an error message and re-open the login page
log out enter a invalid username and invalid password click login button	The application should display an error message and re-open the login page
log out enter a valid username and invalid password click login button	The application should display an error message and re-open the login page

Verify the following username and password combinations.[Note: v for valid and I for invalid]	Note: E means error message and H means Home page
blank, blank	E
blank , i	E
blank, v	E
I, blank	E
I, i	E
I, v	E
I, blank	E
v, i	E
v, v	H

Example 2: Write a test case to test a valid mobile number using manual testing.

Prerequisite: None

Outcome: Students should be able to understand manual testing and creation of test cases to test a valid mobile number.

Test Cases:

c8:mobile number is incorrect .										*	f	f	f
c9: country code is not shown with mobile number field											*	f	f
c10: user can't copy or paste mobile number.												*	f
a1:invalid number recheck				T	T	T	T	T	T	T	T	f	f
a2:valid press enter.												T	T

Test cases	Input	Expected Output
1		invalid number recheck
2	1a11111111	invalid number recheck
3		invalid number recheck
4	1#2#+11111	invalid number recheck
5	1 26626262	invalid number recheck
6	36253111	invalid number recheck
7	14512452141	invalid number recheck
8	1125465214	invalid number recheck
9	4569745651	invalid number recheck
10	9111111111	valid press enter
11	9111111111	valid press enter

Conclusion :Students are able to write test cases .

Experiment No -4

Aim: Introduction to Selenium (WebDriver).

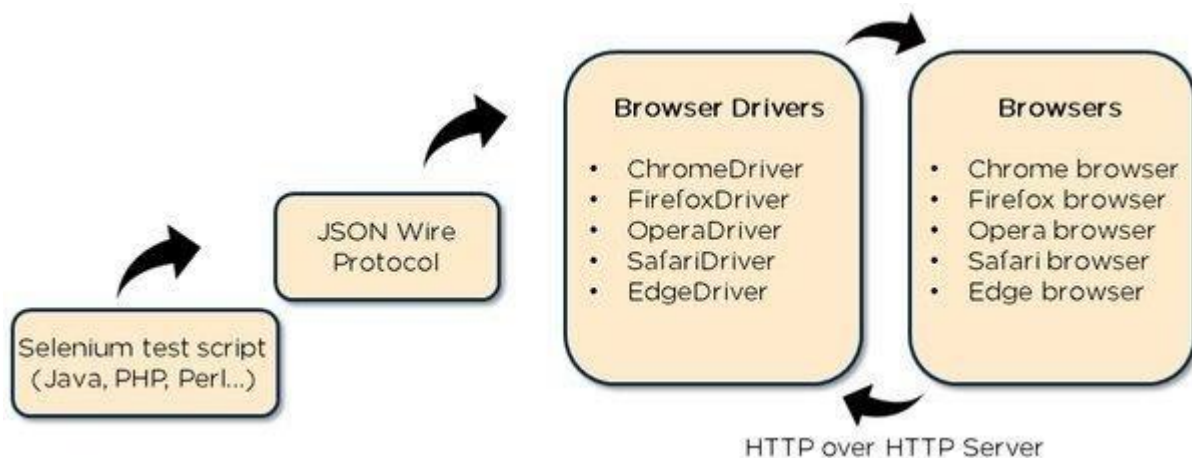
Prerequisite: Basic understanding of Automation Testing.

Outcome: Students should be able to understand how automation testing tool selenium is used.

Theory:

- Paul Hammant developed Selenium WebDriver in 2006. Selenium WebDriver was the first cross-platform testing framework that could configure and control the browsers on the OS level. It served as a programming interface to create and run test cases.
- WebDriver performs actions on web elements. It supports various programming languages like Java, C#, PHP, Python, among others. It can also be integrated with frameworks like TestNG and JUnit for test management.

The Architecture of Selenium WebDriver



- Selenium test script - Selenium test script is the test code written in any of the mentioned programming languages that are interpreted by the driver
- JSON Wire Protocol - JSON Wire Protocol provides a transport mechanism to transfer data between a server and a client. JSON Wire Protocol is the industry standard for various web services
- Browser drivers - Selenium uses drivers, specific to each browser to establish a secure connection with the browser

- Browsers - Selenium WebDriver supports multiple web browsers to test and run applications on.
-

Conclusion :Students are able to understand what is selenium and for what it is being used.

Experiment No -5

Aim: Steps to install Selenium WebDriver.

Prerequisite: Basic understanding of Automation Testing, java and Chrome Web Browser.

Outcome: Students should be able setup Selenium on their local system.

Procedure:

Selenium WebDriver- Installation

Selenium WebDriver installation process is completed in four basic steps:

1. Download and Install Java 8 or higher version.
2. Download and configure Eclipse or any Java IDE of your choice.
3. Download Selenium WebDriver Java Client
4. Configure Selenium WebDriver

1. Download and Install Java

We assume that you have already installed Java 8 or above on your machine and successfully configured the environment variables required to run and compile java programs.

However, you can download the latest version of Java Development Kit (JDK) from the link given below.

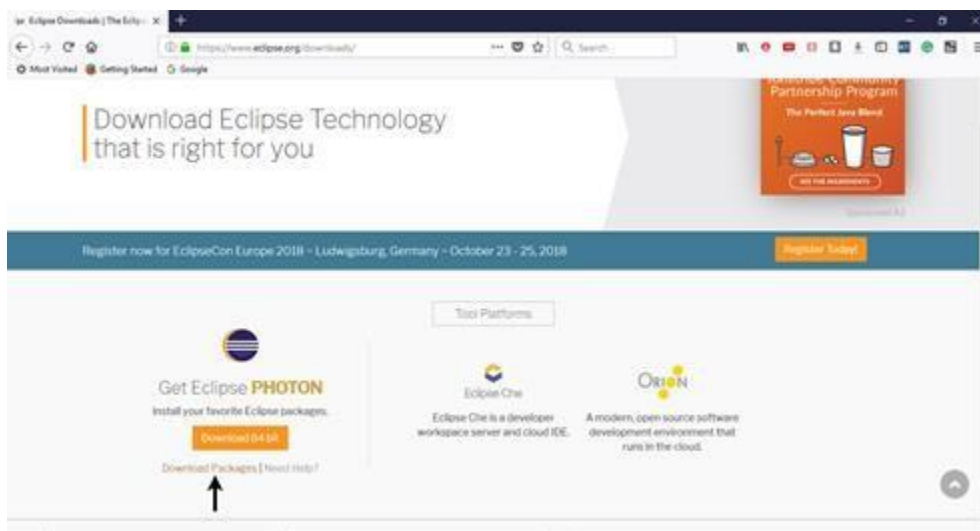
<http://www.oracle.com/technetwork/java/javase/downloads/index.html>

Once you have downloaded and installed the latest version of Java, you need to set path or configure the environment variables in your system. Refer the link given below to understand how we can set path and configure environment variables in Java.

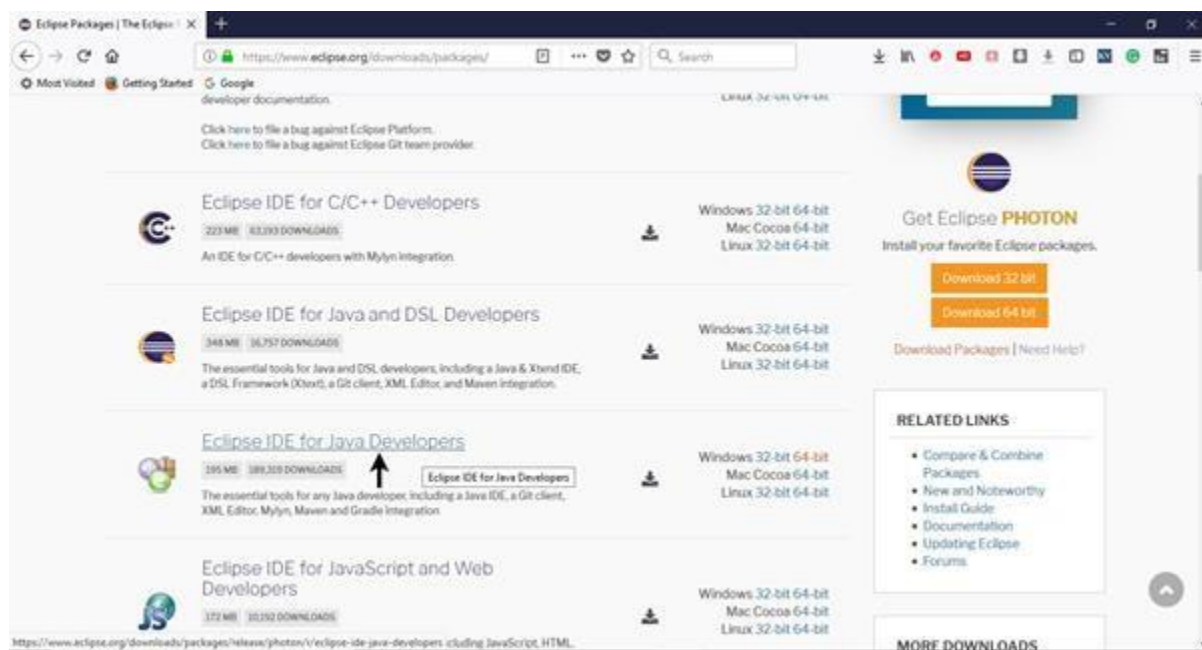
<https://www.javatpoint.com/how-to-set-path-in-java>

2. Download and Configure Eclipse IDE

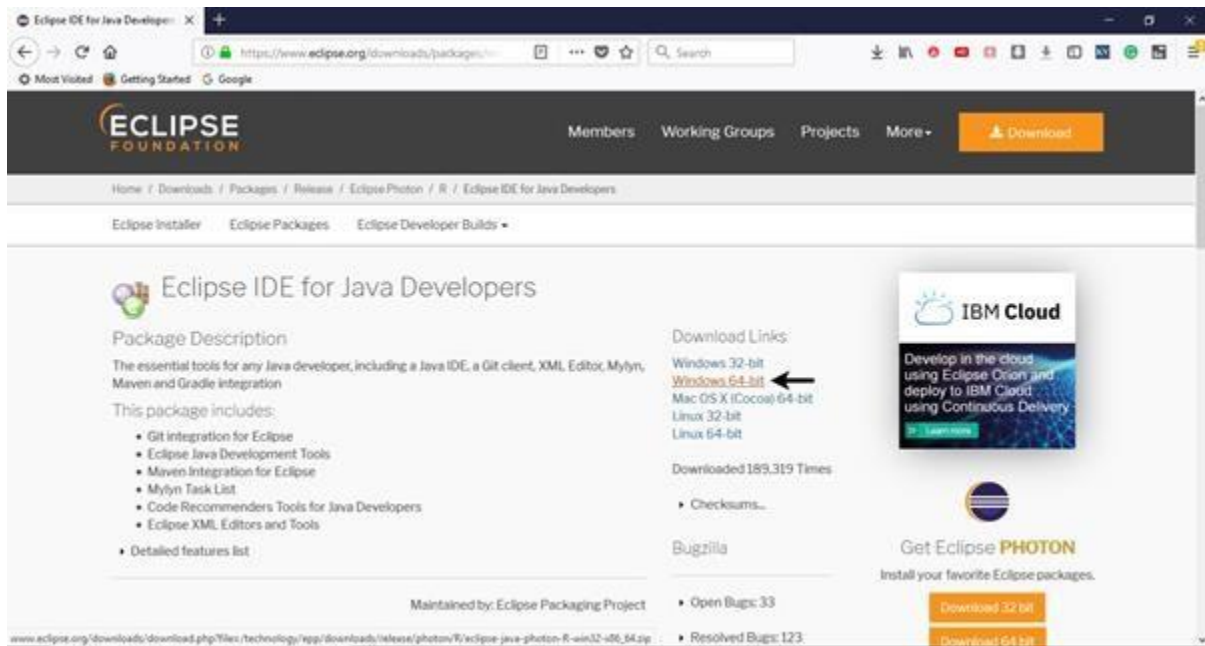
- Open URL: <https://www.eclipse.org/downloads/>.
- Click on the "Download Packages" link (you can also download the IDE directly from the "downloads page" of Eclipse official website, but we will recommend you to navigate through the download packages section and get "Eclipse IDE for Java Developers").



- It will redirect you to the "Download Packages" section. Scroll down through the webpage and click on "Eclipse IDE for Java Developers".



- Go to the Download Links section and click on "Windows 64-bit". You can also select other options to download based on the operating system you are currently working on.



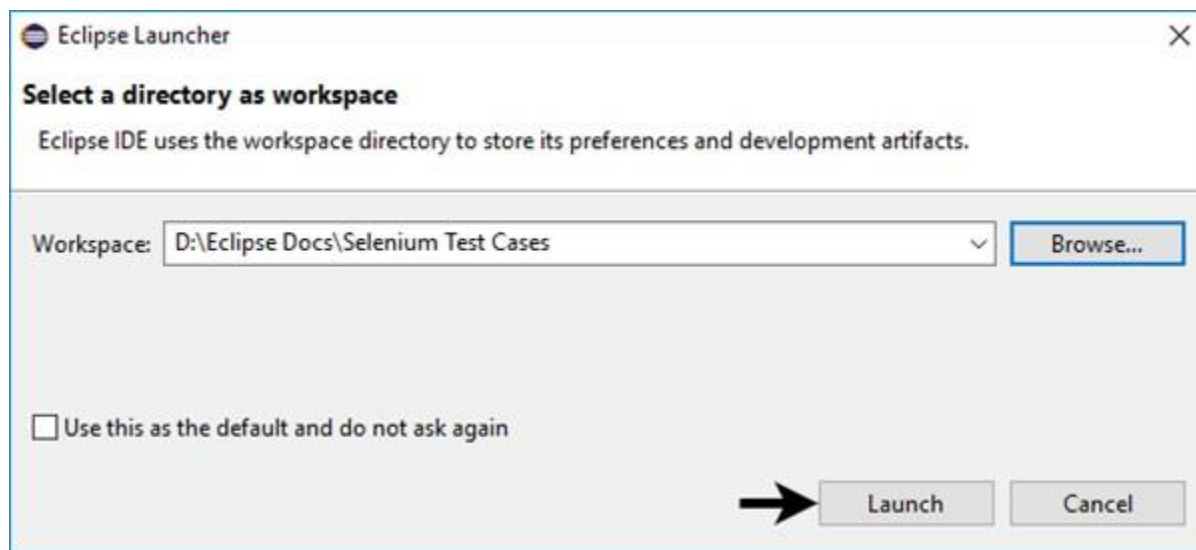
- The downloaded file would be in zipped format. Unpack the contents in a convenient directory.

Name	Date modified	Type	Size
eclipse	20-06-2018 08:09	File folder	
eclipse-java-photon-R-win32-x86_64(1)	02-08-2018 12:44	WinRAR ZIP archive	1,99,756 KB

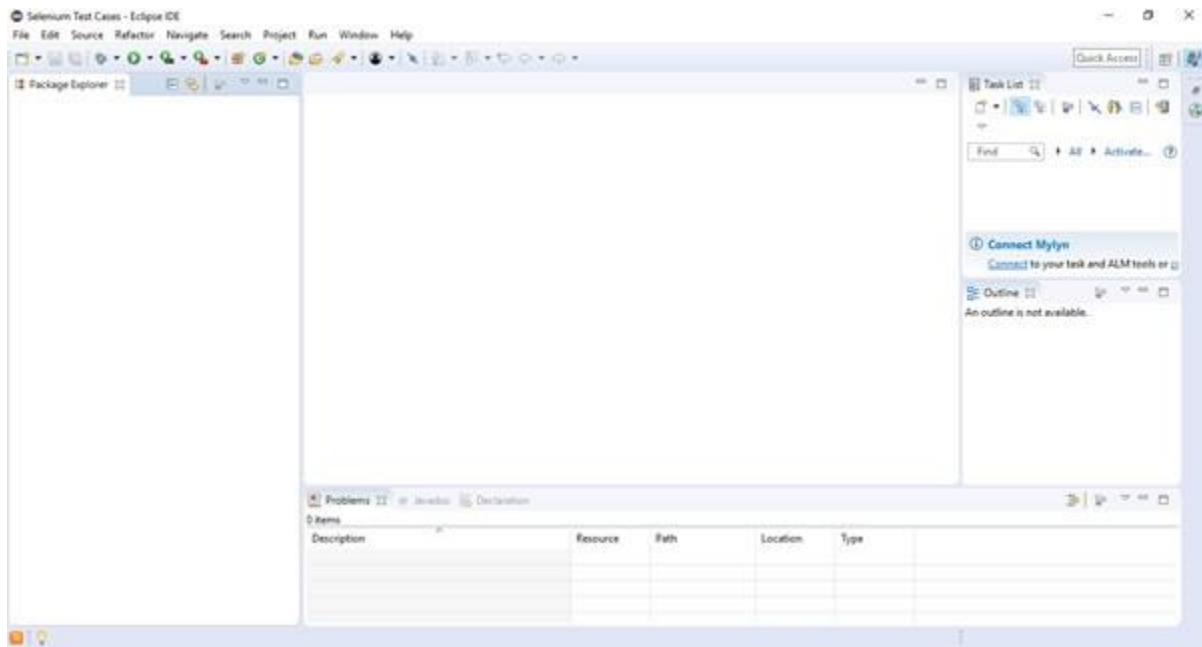
- Double click on "eclipse" (.exe file).

Name	Date modified	Type	Size
configuration	20-06-2018 08:08	File folder	
dropins	20-06-2018 08:08	File folder	
features	20-06-2018 08:08	File folder	
p2	20-06-2018 08:08	File folder	
plugins	20-06-2018 08:08	File folder	
readme	20-06-2018 08:08	File folder	
.eclipseproduct	15-05-2018 10:07	ECLIPSEPRODUCT...	1 KB
artifacts	20-06-2018 08:08	XML Document	140 KB
eclipse	20-06-2018 08:09	Application	415 KB
eclipse	20-06-2018 08:08	Configuration sett...	1 KB
eclipsec	20-06-2018 08:09	Application	127 KB

- To configure the workspace, select a convenient directory where you want to keep all of your Selenium trails and click on Launch button.



- It will launch the default interface of Eclipse IDE.



3. Download Selenium WebDriver Java Client

- Open [URL: https://docs.seleniumhq.org/download/](https://docs.seleniumhq.org/download/)
It will redirect you to the "downloads page" of Selenium official website.
- Scroll down through the web page and locate **Selenium Client & WebDriver Language Bindings**.
- Click on the "Download" link of Java Client Driver as shown in the image given below.








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	3.13.0	2018-06-25	Download	Change log	Javadoc
C#	3.13.0	2018-06-25	Download	Change log	API docs
Ruby	3.13.1	2018-07-20	Download	Change log	API docs
Python	3.13.0	2018-06-25	Download	Change log	API docs
Javascript (Node)	4.0.0-alpha.1	2018-01-13	Download	Change log	API docs

- The downloaded file would be in zipped format. Unpack the contents in a convenient directory. It contains the essential jar files required to configure Selenium WebDriver in Eclipse IDE.

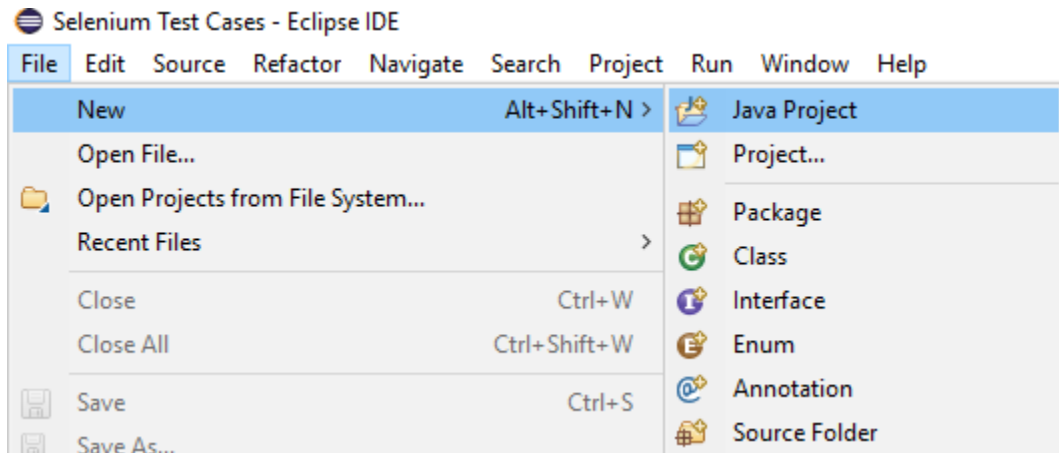
Name	Date modified	Type	Size
 libs	01-02-1985 12:00	File folder	
 CHANGELOG	01-02-1985 12:00	File	116 KB
 client-combined-3.13.0	01-02-1985 12:00	Executable Jar File	1,482 KB
 client-combined-3.13.0-sources	01-02-1985 12:00	Executable Jar File	508 KB
 LICENSE	01-02-1985 12:00	File	12 KB
 NOTICE	01-02-1985 12:00	File	1 KB
 selenium-java-3.13.0(1)	03-08-2018 04:03	WinRAR ZIP archive	8,847 KB

4. Configure Selenium WebDriver

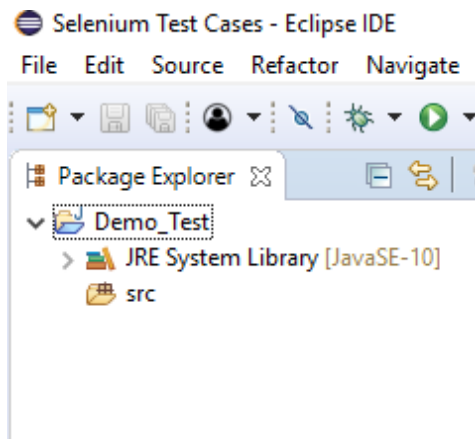
Now we will configure our Eclipse IDE with Selenium WebDriver. In simple terms, we will create a new Java Project in Eclipse and load all the essential jar files in order to create Selenium Test Scripts.

- Launch Eclipse IDE

- Create a new Java Project from **File > New > Java Project**.

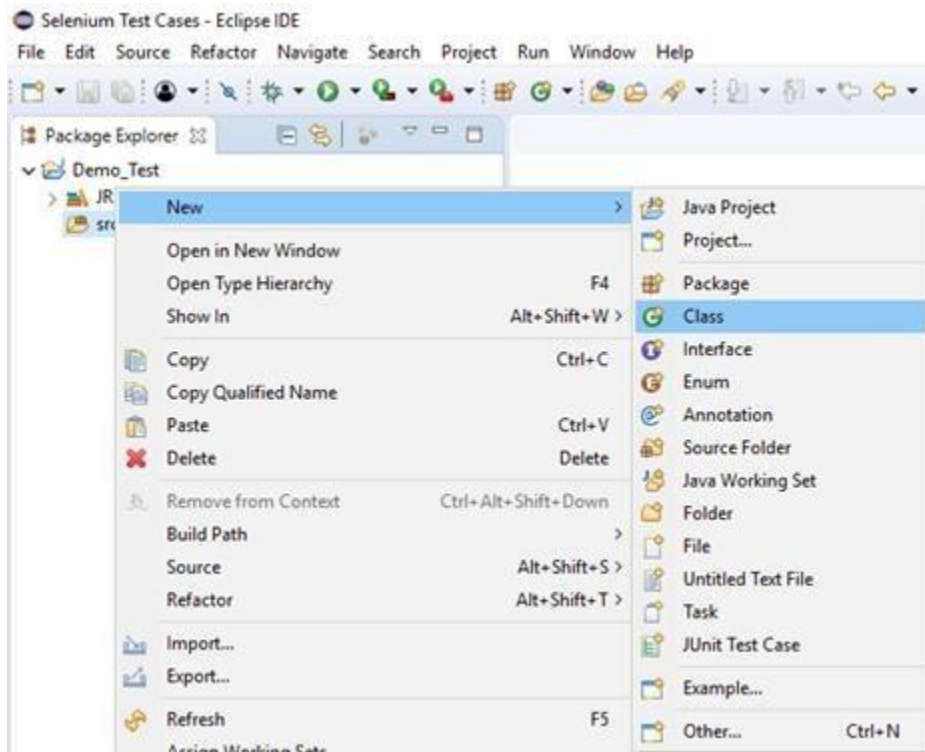


- Give your Project name as "Demo_Test", leave the other fields unaltered and click on "Finish" button.
- It will create a new Java project with the following directories.

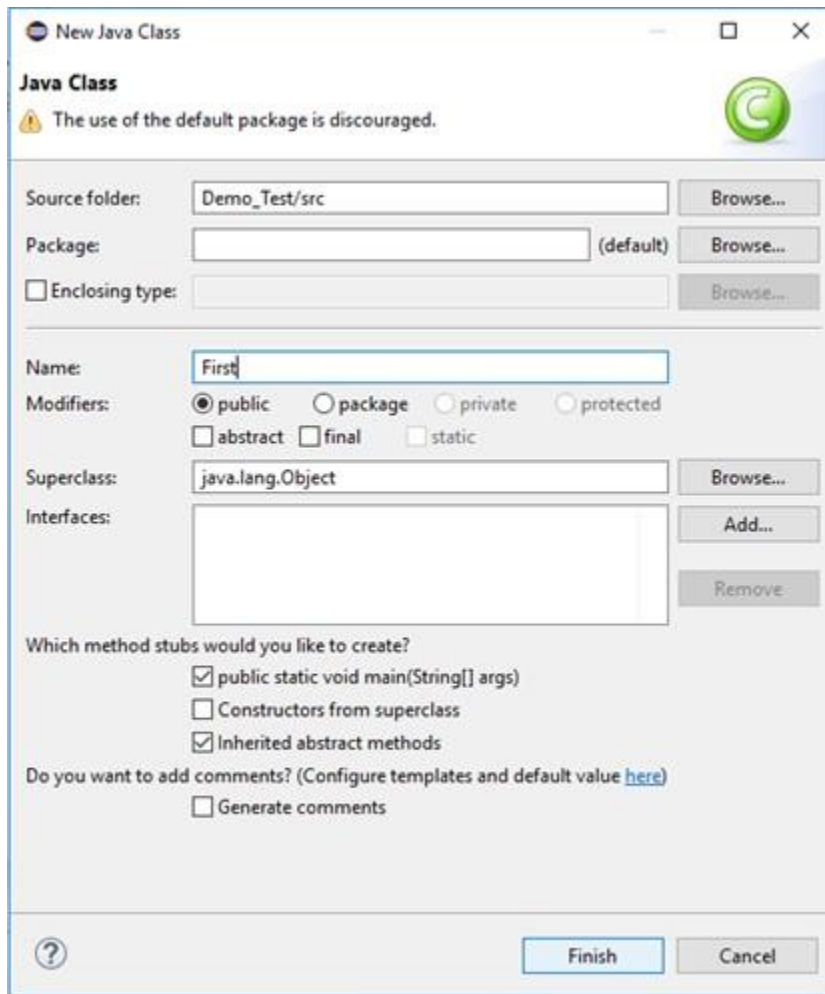


Note: Selenium Test Scripts are always written in ".class" file in Java. Here the project "Demo_Test" act as a Test Suite that may contain one or more Selenium test cases/test scripts.

- Right click on the "src" folder and create a new Class File from **New > Class**.

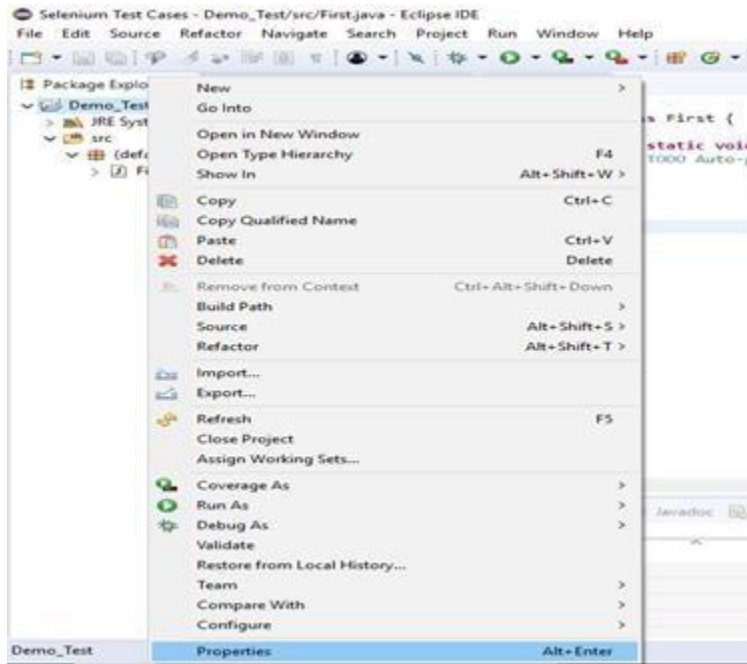


- Give your Class name as "First" and click on "Finish" button.

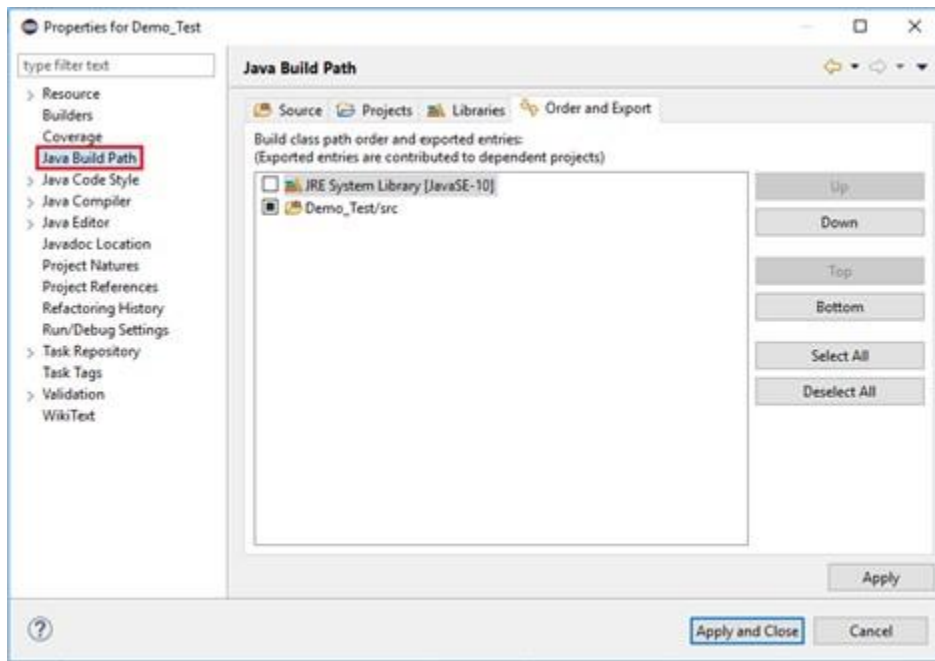


Now, we will add the Selenium jar files in our Test Suite (Demo_Test).

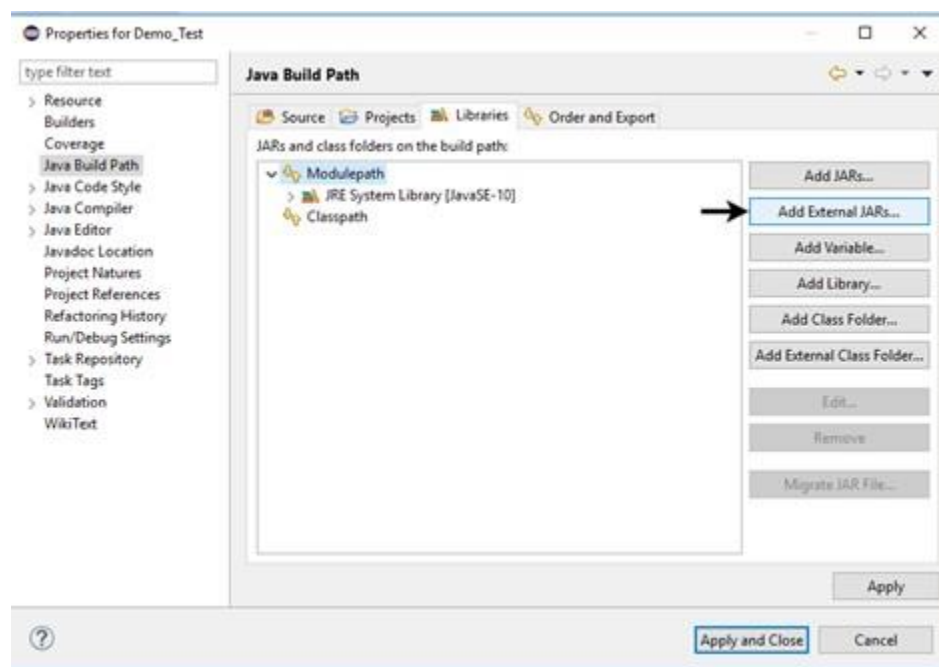
- Right click on "Demo_Test" folder and select Properties.



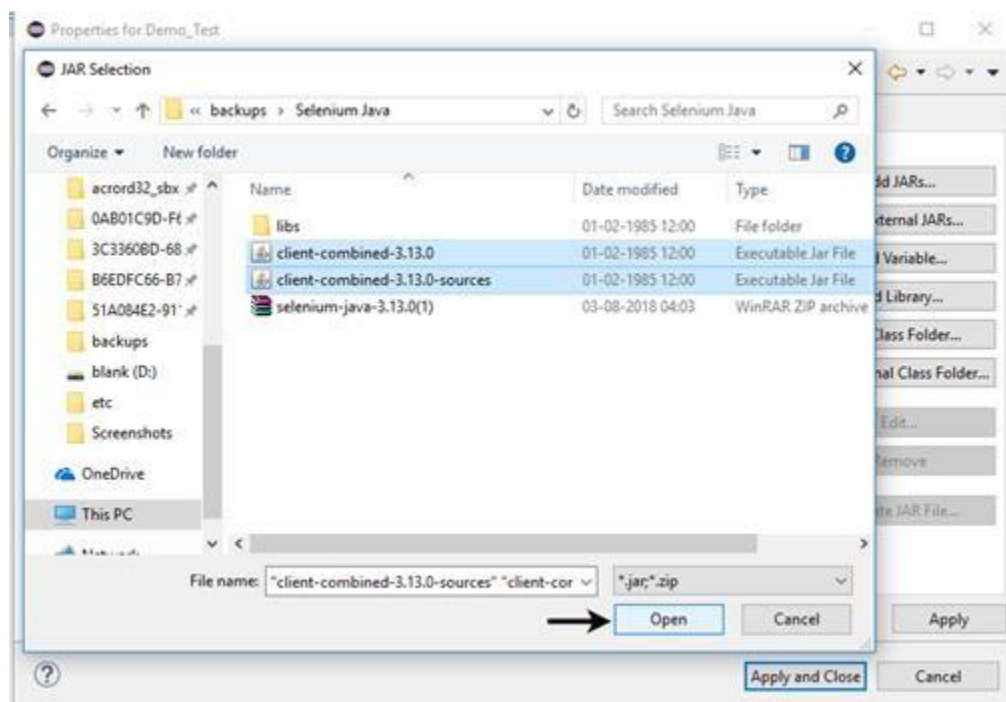
- It will launch the Properties window for our "Demo_Test" Test Suite.
- Click on "Java Build Path" option from the left hand side panel.



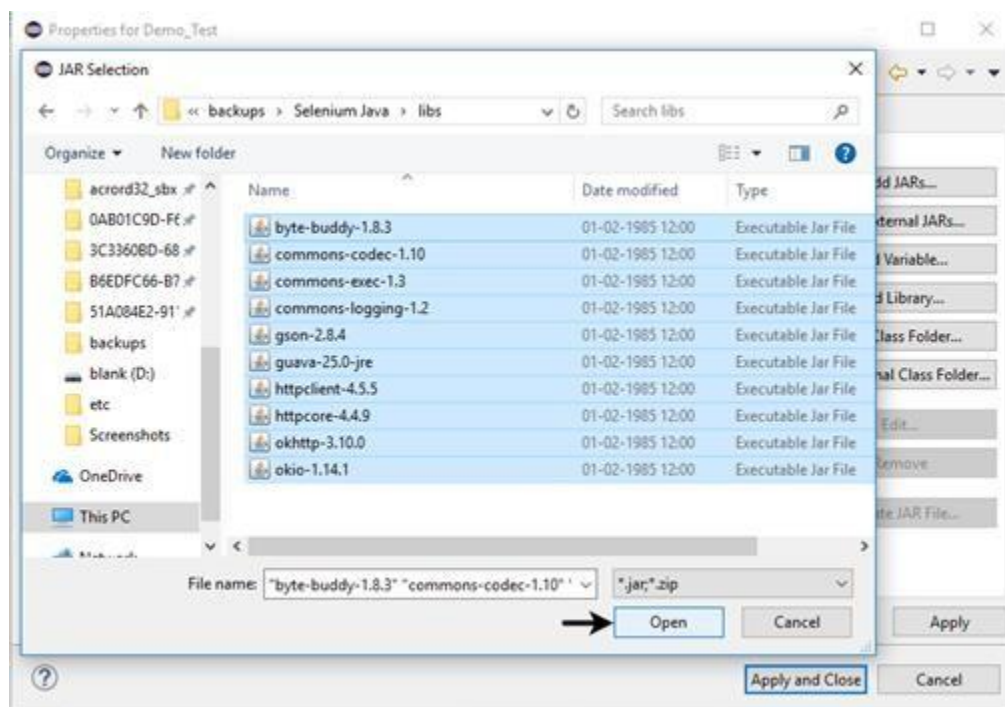
- Switch to Libraries tab and click on "Add External JARs" button.



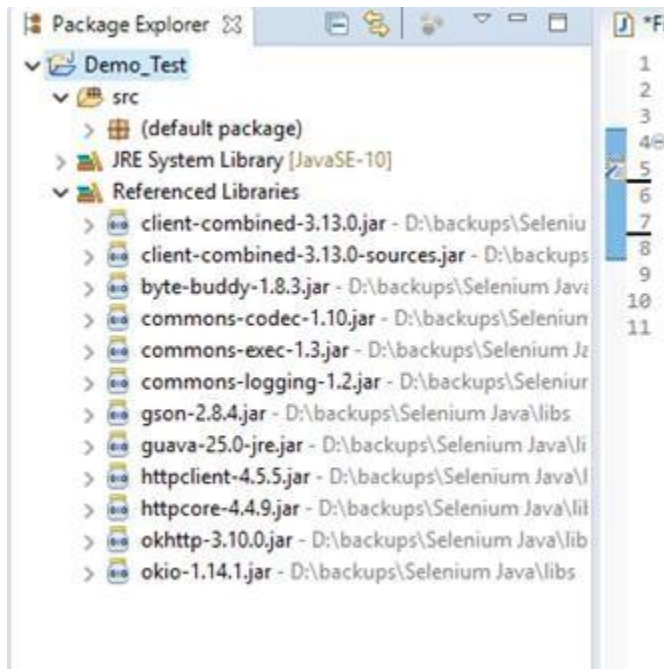
- Locate the directory where you have downloaded the Selenium jar files, select the respective jars and click on "Open" button.



- Repeat the same steps for the jars which are present under the "libs" folder.
- Open "libs" folder, select all of the respective jar files and click on "Open" button.



- Once you get all the Selenium jar files in your Libraries tab, click on Apply and Close button.
- The following image shows the directory structure of our "Demo_Test" test suite after adding Selenium jars.



Hence, we have successfully configured Selenium WebDriver with Eclipse IDE. Now, we are ready to write our test scripts in Eclipse and run it in WebDriver.

Conclusion: Student are able to install Selenium WebDriver.

Experiment No -6

Aim: Selenium WebDriver- First Test Case

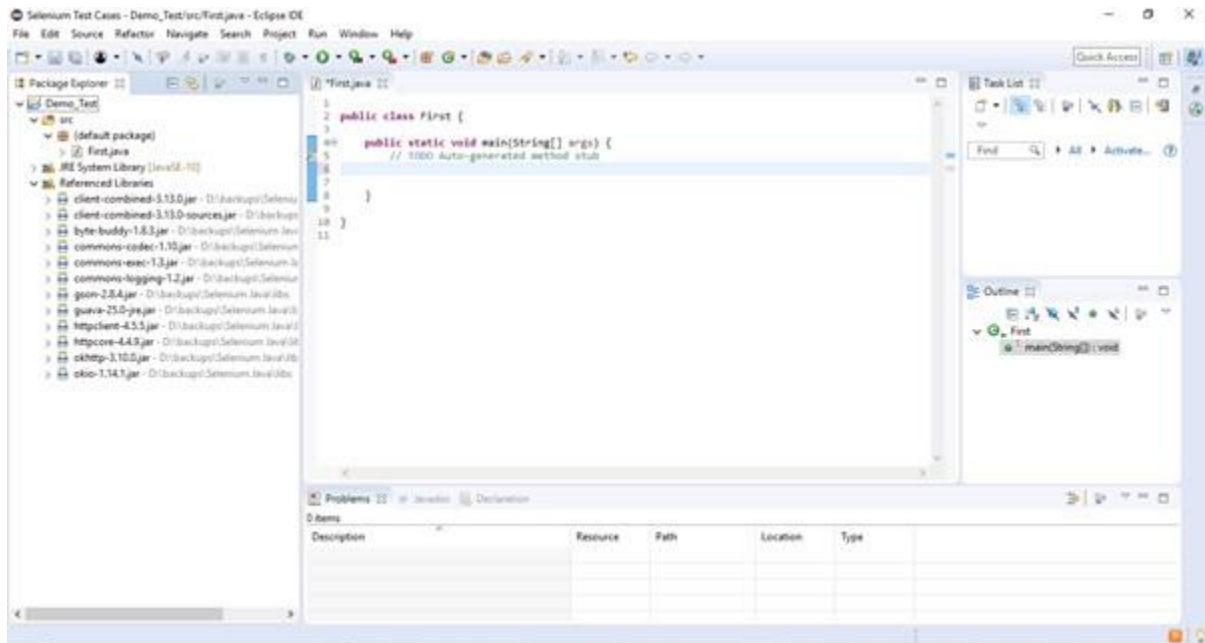
In this section, you will learn how to create your First Selenium Automation Test Script.

Under this test, we will automate the following scenarios:

- Invoke Google Chrome browser.
- Open URL: www.google.com
- Click on the Google Search text box.
- Type the value "javatpoint tutorials"
- Click on the Search button.

We will create our test case step by step to give you a complete understanding of each component in detail.

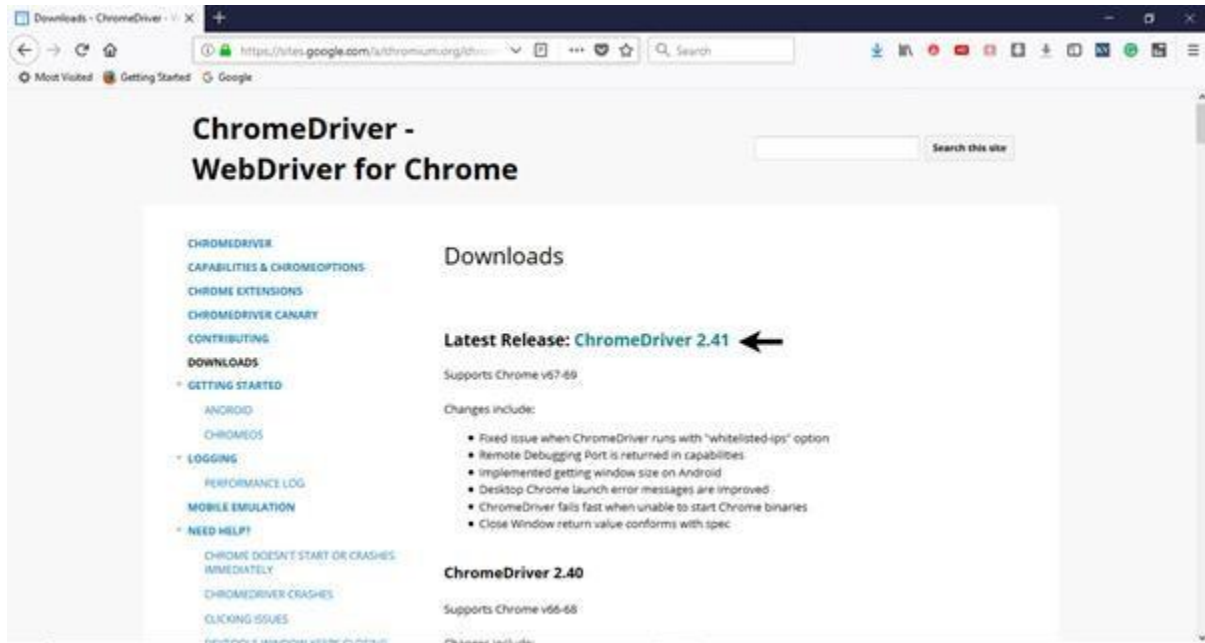
Step1. Launch Eclipse IDE and open project "Demo_Test" which we have created in the previous section (Configure Selenium WebDriver) of this Tutorial. We will write our first Selenium test script in the "First.class" file under the "Demo_Test" test suite.



Note: To invoke a browser in Selenium, we have to download an executable file specific to that browser. For example, Chrome browser implements the WebDriver protocol using an executable called ChromeDriver.exe. These executable files start a server on your system which in turn is responsible for running your test scripts

Step2. Open URL: <https://sites.google.com/a/chromium.org/chromedriver/downloads> in your browser.

Step3. Click on the "ChromeDriver 2.41" link. It will redirect you to the directory of ChromeDriver executables files. Download as per the operating system you are currently on.



For windows, click on the "chromedriver_win32.zip" download.

Index of /2.41/

Name	Last modified	Size	ETag
Parent Directory	-	-	-
chromedriver_linux64.zip	2018-07-27 19:25:01	3.76MB	fbd8b9561575054e0e7e9cc53b680a70
chromedriver_mac64.zip	2018-07-27 20:45:35	5.49MB	4c86429625373392bd9773c9d0a1c6a4
chromedriver_win32.zip	2018-07-27 21:44:20	3.39MB	ab047aa361aeb863e58514a9f46bcd7
notes.txt	2018-07-27 21:58:29	0.02MB	0b595efd8eec0ed4352c69bba64e0d7c

The downloaded file would be in zipped format. Unpack the contents in a convenient directory.

Name	Date modified	Type	Size
 chromedriver	27-07-2018 12:32	Application	6,580 KB
 chromedriver_win32	10-08-2018 11:31	WinRAR ZIP archive	3,469 KB

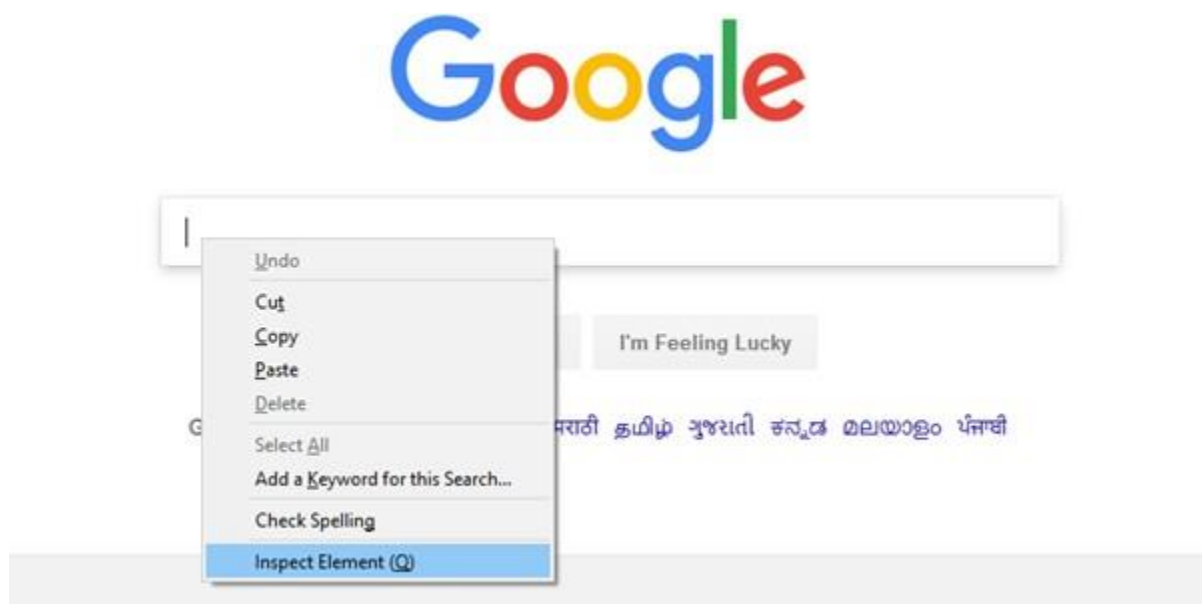
Note: Selenium developers have defined properties for each browser that needs the location of the respective executable files to be parsed in order to invoke a browser. For example, the property defined for Chrome browser - `webdriver.chrome.driver`, needs the path of its executable file - `D:\ChromeDriver\chromedriver.exe` in order to launch chrome browser.

```
System.setProperty("webdriver.chrome.driver", "D:\\ChromeDriver\\chromedriver.exe");
```

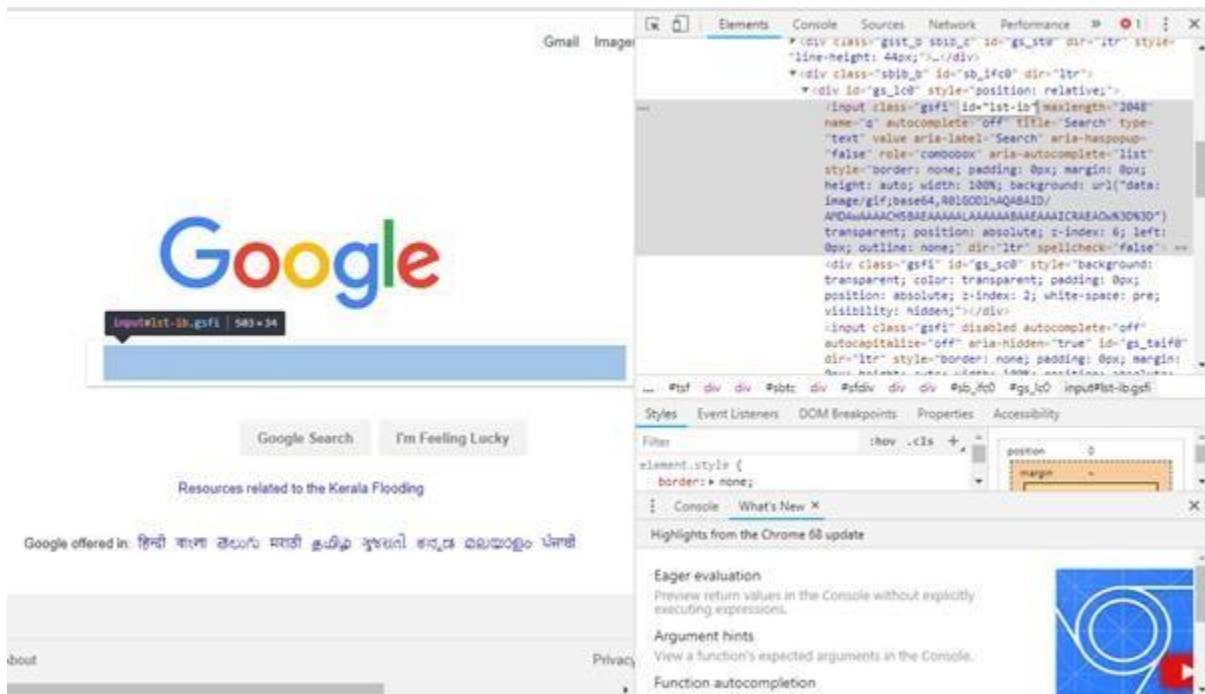
Step4. We would need a unique identification for the web elements like Google Search text box and Search button in order to automate them through our test script. These unique identifications are configured along with some Commands/Syntax to form Locators. Locators help us to locate and identify a particular web element in context of a web application.

The method for finding a unique identification element involves inspection of HTML codes.

- Open URL: <https://www.google.com> in your Chrome browser.
- Right click on the Google search text box and select Inspect Element.



- It will launch a window containing all the specific codes involved in the development of the test box.



- Pick the value of id element i.e. "lst-ib".

```
<input class="gsfi" id="lst-ib" maxlength="2048"
name="q" autocomplete="off" title="Search" type="
```

- Given below is the Java syntax for locating elements through "id" in Selenium WebDriver.

```
driver.findElement(By.id (<element ID>))
```

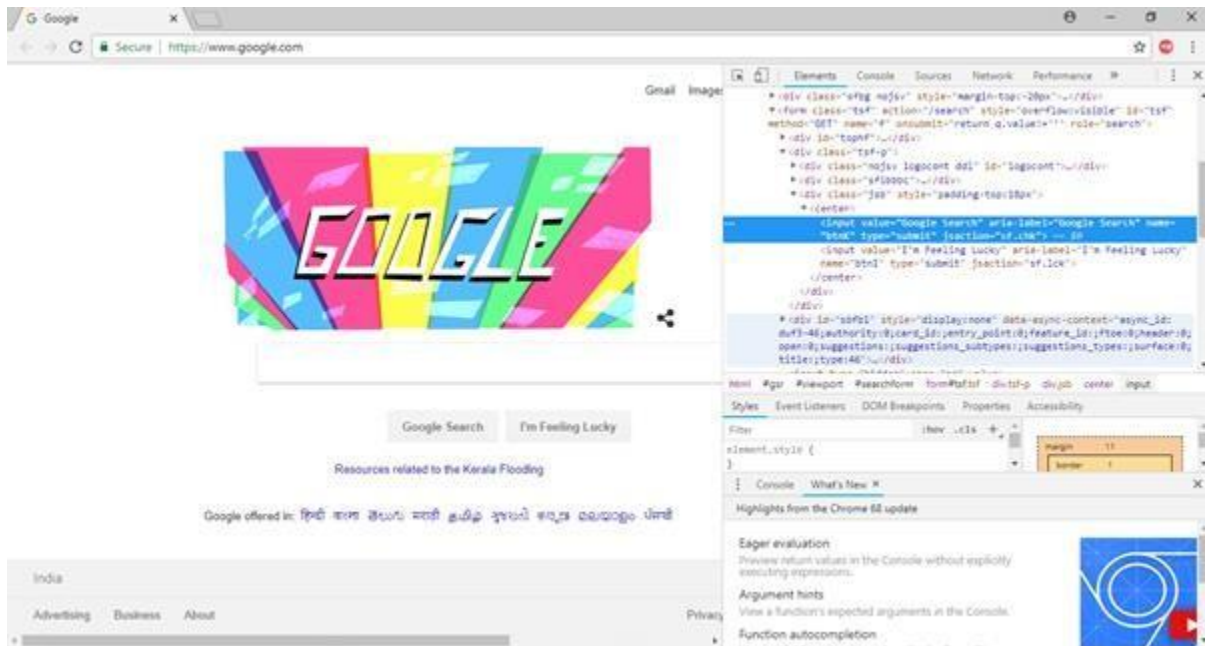
- Here is the complete code for locating Google Search text box in our test script.

```
driver.findElement(By.id ("lst-ib"))
```

- Now, right click on the Google Search button and select Inspect Element.



- It will launch a window containing all the specific codes involved in the development of the Google Search button.



- Pick the value of **name** element i.e. "btnK".

```
<input value="Google Search" aria-label="Google Search"
name="btnK" type="submit" jsaction="sf.chk"> == $0
```

- Given below is the Java syntax for locating elements through "name" in Selenium WebDriver.

```
driver.findElement(By.name (<element name>))
```

- Here is the complete code for locating Google Search button in our test script.

```
driver.findElement(By.name ("btnK"))
```

Step5. Now it is time to code. We have embedded comments for each block of code to explain the steps clearly.

Conclusion:

Students are able to launch the browser using selenium WebDriver.

Experiment No -7

Aim: Selenium WebDriver- Commands

Selenium commands are the set of commands that are used to run our Selenium tests.

In Selenium WebDriver, we have an entirely different set of commands for performing different operations. Since we are using Selenium WebDriver with Java, commands are simply **methods** written in Java language.

Differents Commands in Selenium

1. Fetching a web page

There are two methods to fetch a web page:

- Using Get method

```
driver.get("www.google.com")
```

- Using Navigate method

```
driver.navigate().to("https://www.google.com/");
```

2. Locating forms and sending user inputs

1. `driver.findElement(By.id("lst-ib")).sendKeys("javatpoint tutorials");`

3. Clearing User inputs

The `clear()` method is used to clear the user inputs from the text box.

```
driver.findElement(By.name("q")).clear();
```

4. Fetching data over any web element

Sometimes we need to fetch the text written over a web element for performing some assertions and debugging. We use `getText()` method to fetch data written over any web element.

```
driver.findElement(By.id("element567")).getText();
```

5. Performing Click event

The `click()` method is used to perform click operation on any web element.

```
driver.findElement(By.id("btnK")).click();
```

6. Navigating backward in browser history

```
driver.navigate().back();
```

7. Navigating forward in browser history

```
driver.navigate().forward();
```

8. Refresh/ Reload a web page

```
driver.navigate().refresh();
```

9. Closing Browser

```
driver.close();
```

10. Closing Browser and other all other windows associated with the driver

```
driver.quit();
```

11. Moving between Windows

```
driver.switchTo().window("windowName");
```

13. Moving between Frames

```
driver.switchTo().frame("frameName");
```

14. Drag and Drop

Drag and Drop operation is performed using the Action class.

1. `WebElement element = driver.findElement(By.name("source"));`
2. `WebElement target = driver.findElement(By.name("target"));`
3. `(new Actions(driver)).dragAndDrop(element, target).perform();`

Conclusion : Students are able to implement all selenium WebDriver Commands.

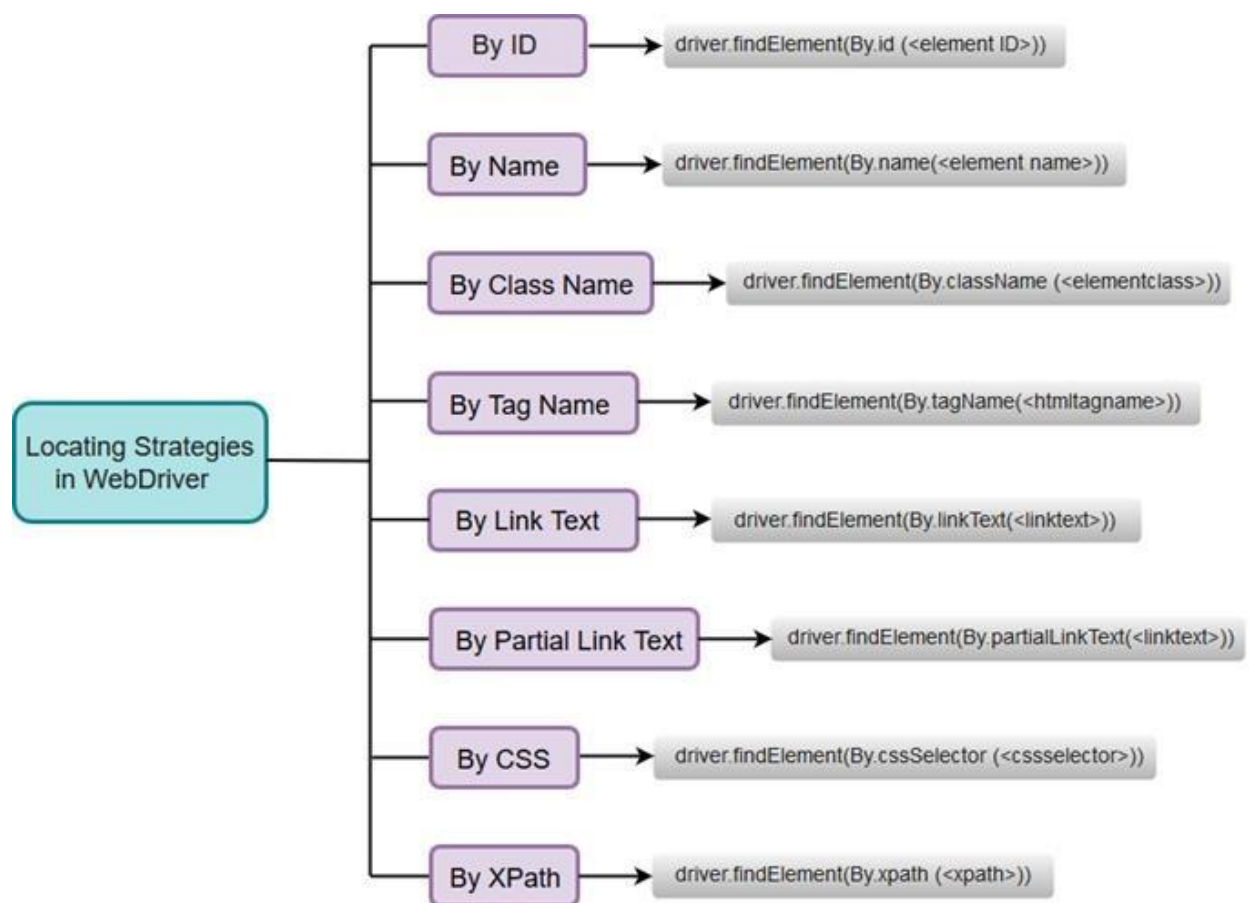
Experiment No -8

Aim: Understand the different types of Locator in Selenium WebDriver

we are using WebDriver with java; each locating strategy has its own command in Java to locate the web elements.

Note: Locating web elements in Webdriver is performed with the help of `findElement()` and `findElements()` method.

A list of Locating Strategies used in WebDriver:



- [Locating Strategies By ID](#)

The Java Syntax for locating a web element using its id attribute is written as:

```
driver.findElement(By.id (<element ID>))
```

- Locating Strategies By Name

The Java Syntax for locating a web element using its name attribute is written as:

```
driver.findElement(By.name(<element ID>))
```

- Locating Strategies By Class Name

The Java Syntax for locating a web element using its Class attribute is written as:

```
driver.findElement(By.className (<element class
```

- Locating Strategies By Tag Name

The Java Syntax for locating a web element using its Tag Name is written as:

```
driver.findElement(By.tagName (<htmltagname>))
```

- Locating Strategies By Link Text

The Java Syntax for locating a web element through its Link Text is written as:

```
driver.findElement(By.linkText (<linktext>))
```

- Locating Strategies By Partial Link Text

The Java Syntax for locating a web element through its Partial Link Text is written as:

```
driver.findElement(By.partialLinkText (<linktext>))
```

- Locating Strategies By CSS

The Java Syntax for locating a web element through CSS - Tag and ID Selector is written as:

```
driver.findElement(By.cssSelector("Tag#Value of id attribute"))
```

- Locating Strategies By XPath

In WebDriver, the Java syntax for locating elements through XPath can be written as:

```
findElement(By.xpath("XPath"));
```

Conclusion: Students are able to understand to understand different types of Locators in selenium.

Experiment No -9

Aim: Selenium WebDriver- Handling Alerts.

Selenium WebDriver provides three methods to accept and reject the Alert depending on the Alert types.

1. void dismiss()

This method is used to click on the 'Cancel' button of the alert.

Syntax: driver.switchTo().alert().dismiss();

2. void accept()

This method is used to click on the 'Ok' button of the alert.

Syntax:

driver.switchTo().alert().accept();

3. String getText()

This method is used to capture the alert message.

Syntax:

driver.switchTo().alert().getText();

4. void sendKeys(String stringToSend)

This method is used to send some data to the alert box.

Syntax:

1. `driver.switchTo().alert().sendKeys("Text");`

Conclusion: Students are able to understand to Handle Alerts.

Experiment No -10

Aim : Mini-projects

Theory :

Select any one topic and do Mini-project

Write code of project

Conclusion:
