Selenium automates browsers. That's it! What you do with that power is entirely up to you. Primarily it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should) also be automated as well.

## ****What is Selenium?**** Selenium is a free, open-source automation testing tool used to automate web applications for testing purposes. It allows testers and developers to write scripts that simulate user interactions with a web browser — like clicking buttons, filling forms, navigating pages, etc.

It supports multiple:

* **Browsers:** Chrome, Firefox, Edge, Safari, etc.
* **Languages:** Java, Python, C#, JavaScript, Ruby, Kotlin, etc.
* **Operating Systems:** Windows, macOS, Linux.

## ****Why Do We Need Selenium?****

We need Selenium mainly for **automation testing** of web applications. Here’s why it’s so useful:

1. **Saves Time & Effort**
   * Repetitive manual testing (e.g., regression tests) is automated, freeing testers for more complex tasks.
2. **Improves Accuracy**
   * Automation eliminates human errors that often occur in manual testing.
3. **Supports Continuous Integration (CI/CD)**
   * Selenium can be integrated into tools like Jenkins, GitHub Actions, or Azure DevOps to run automated tests after every code change.
4. **Cross-Browser Testing**
   * You can test your application across multiple browsers automatically.
5. **Parallel Test Execution**
   * Selenium Grid lets you run tests on multiple machines/browsers simultaneously — reducing test execution time.
6. **Open Source & Flexible**
   * No licensing cost and easy to customize for various testing needs.

## ****Selenium components****

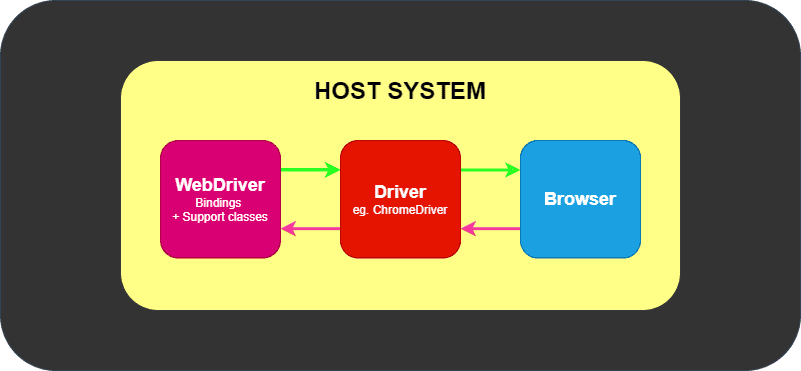
Building a test suite using WebDriver will require you to understand and effectively use several components. As with everything in software, different people use different terms for the same idea. Below is a breakdown of how terms are used in this description.

## ****Terminology****

* **API:** Application Programming Interface. This is the set of “commands” you use to manipulate WebDriver.
* **Library:** A code module that contains the APIs and the code necessary to implement them. Libraries are specific to each language binding, eg .jar files for Java, .dll files for .NET, etc.
* **Driver:** Responsible for controlling the actual browser. Most drivers are created by the browser vendors themselves. Drivers are generally executable modules that run on the system with the browser itself, not the system executing the test suite. (Although those may be the same system.) NOTE: *Some people refer to the drivers as proxies.*
* **Framework:** An additional library that is used as a support for WebDriver suites. These frameworks may be test frameworks such as JUnit or NUnit. They may also be frameworks supporting natural language features such as Cucumber or Robotium. Frameworks may also be written and used for tasks such as manipulating or configuring the system under test, data creation, test oracles, etc.

### The Parts and Pieces

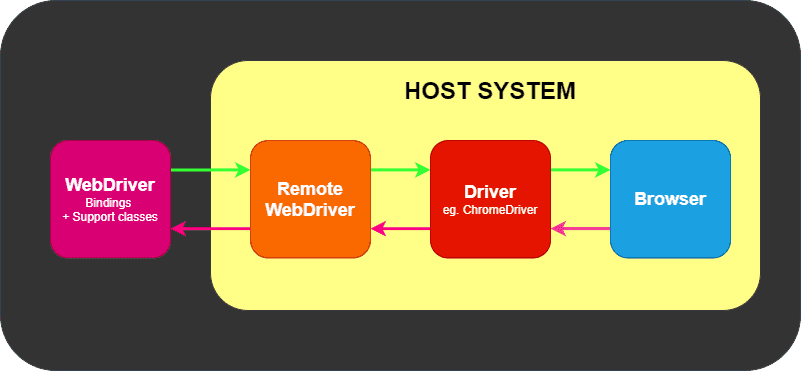
At its minimum, WebDriver talks to a browser through a driver. Communication is two-way: WebDriver passes commands to the browser through the driver, and receives information back via the same route.



**Figure 1:** Web Driver and Browser Communication

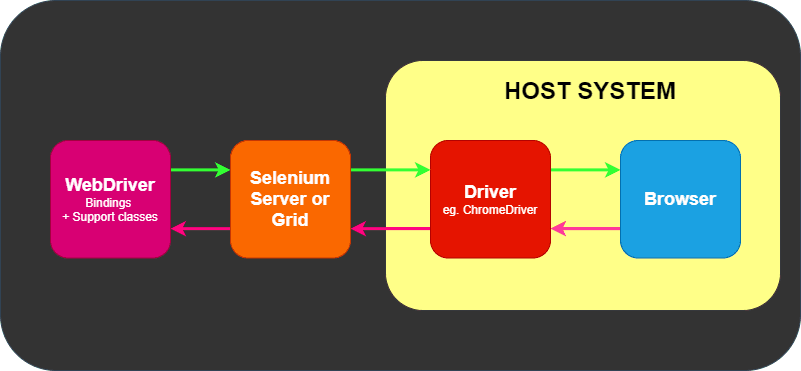
The driver is specific to the browser, such as ChromeDriver for Google’s Chrome/Chromium, GeckoDriver for Mozilla’s Firefox, etc. The driver runs on the same system as the browser. This may or may not be the same system where the tests themselves are executed.

This simple example above is *direct* communication. Communication to the browser may also be *remote* communication through Selenium Server or RemoteWebDriver. RemoteWebDriver runs on the same system as the driver and the browser.



**Figure 2:** Web Driver and Browser Remote Communication

Remote communication can also take place using Selenium Server or Selenium Grid, both of which in turn talk to the driver on the host system

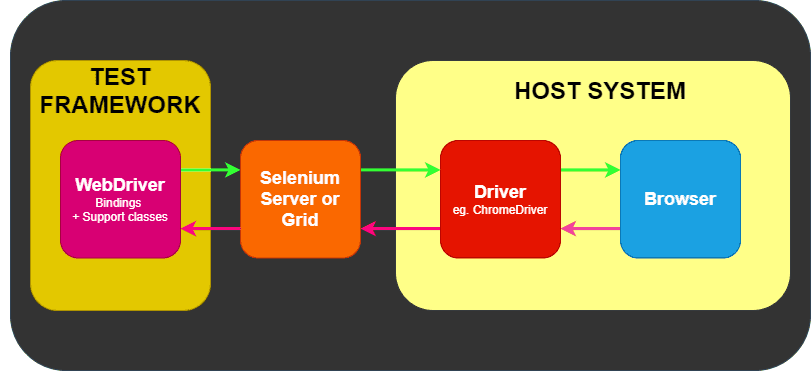


**Where Frameworks fit in**

WebDriver has one job and one job only: communicate with the browser via any of the methods above. WebDriver does not know a thing about testing: it does not know how to compare things, assert pass or fail, and it certainly does not know a thing about reporting or Given/When/Then grammar.

This is where various frameworks come into play. At a minimum, you will need a test framework that matches the language bindings, e.g., NUnit for .NET, JUnit for Java, RSpec for Ruby, etc.

The test framework is responsible for running and executing your WebDriver and related steps in your tests. As such, you can think of it looking akin to the following image.



Natural language frameworks/tools such as Cucumber may exist as part of that Test Framework box in the figure above, or they may wrap the Test Framework entirely in their custom implementation.

**Example Code**

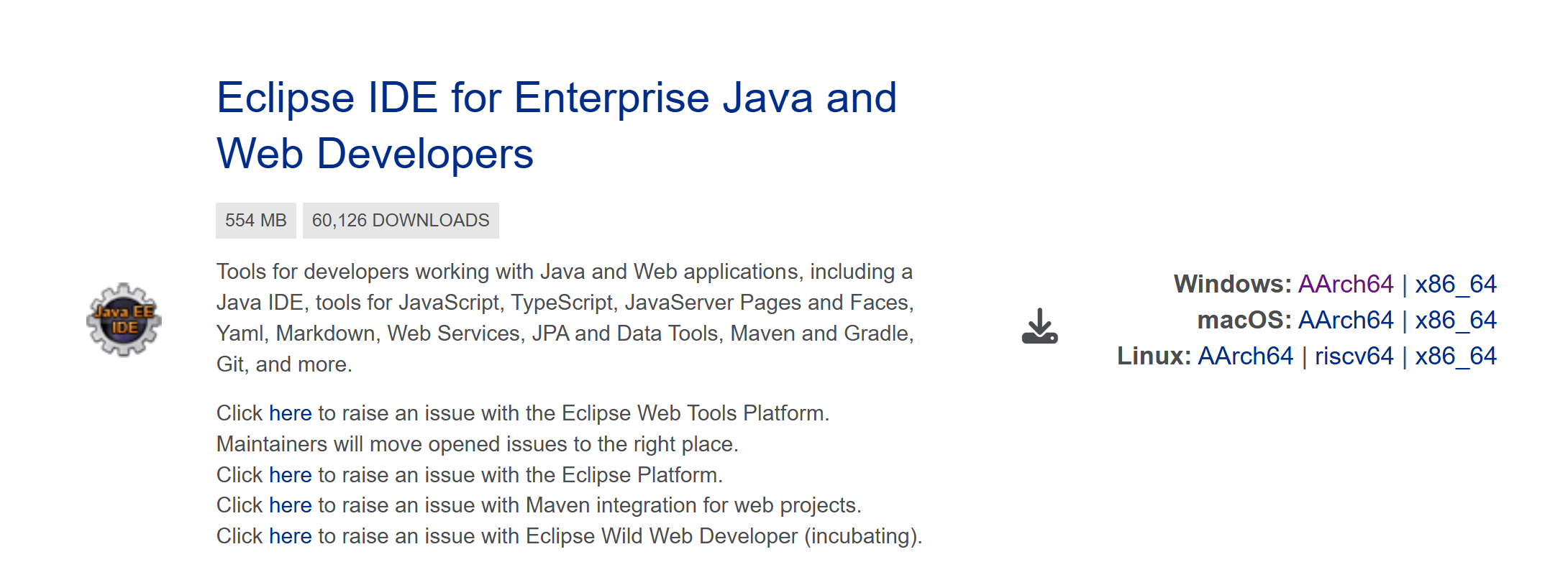
**Installation Steps:**

The following are the steps to download Eclipse IDE which are as follows:

1. **Go to :** [**https://eclipseide.org/**](https://eclipseide.org/)
2. **Click on the following button as shown in the following figure:**

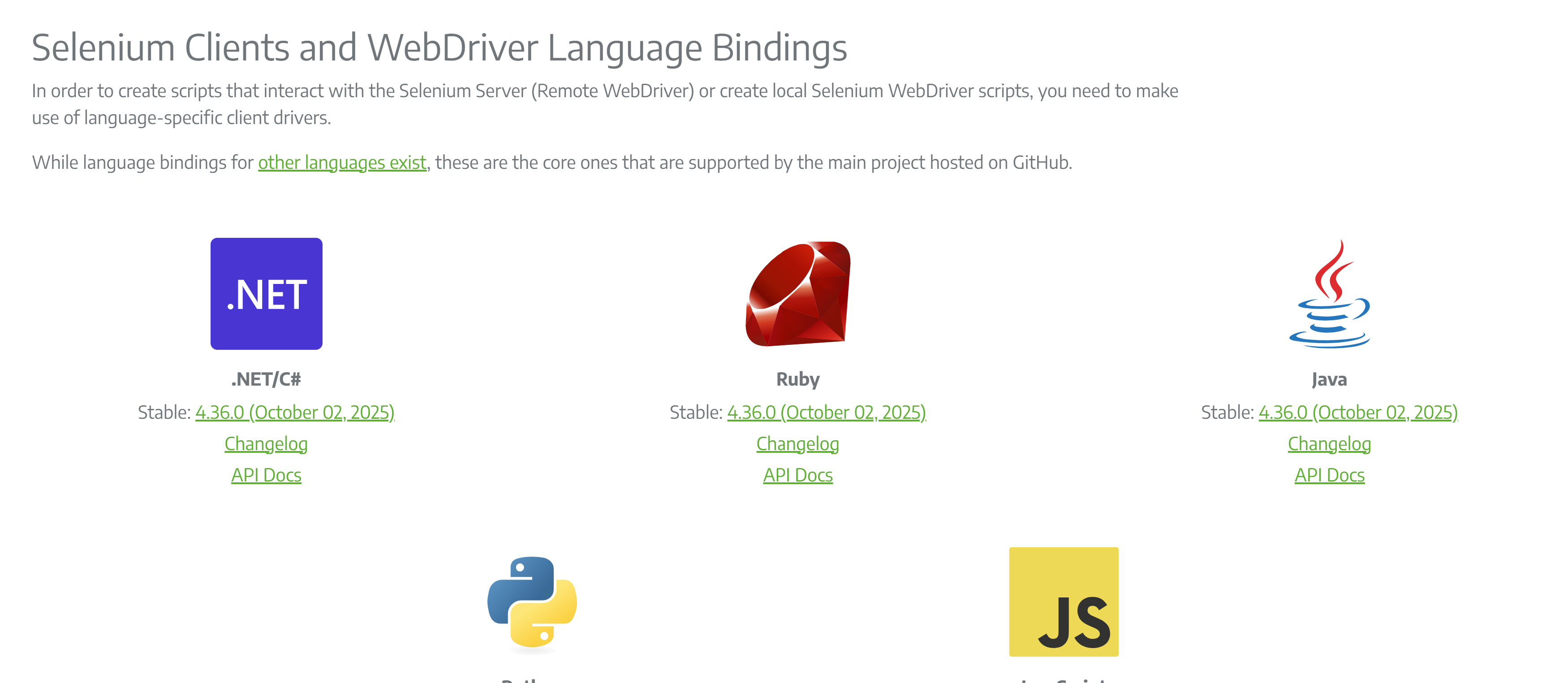
****

1. **Download Eclipse IDE for Enterprise Java and Web Developers**

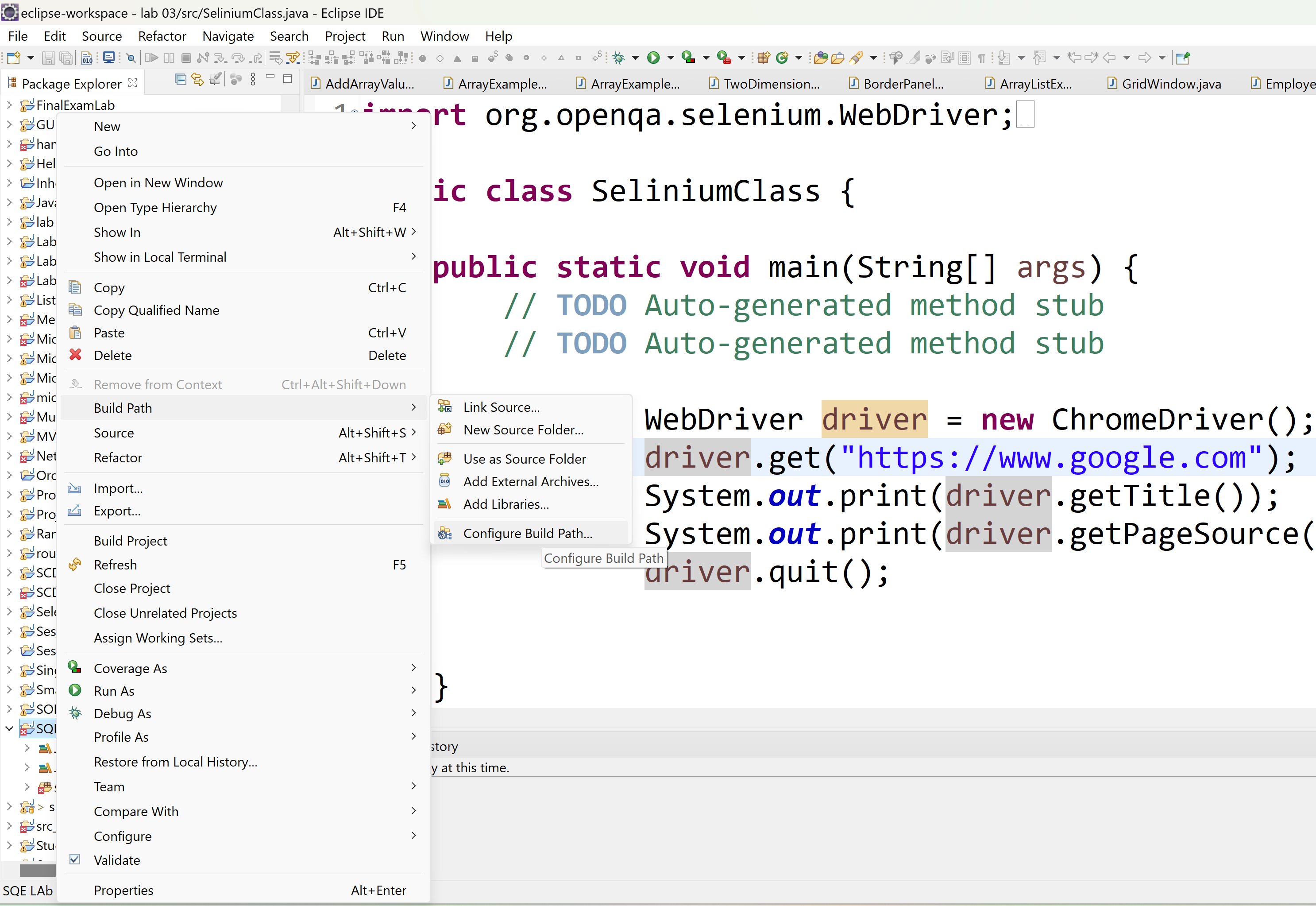
****

The following are the steps to download Selenium which are as follows:

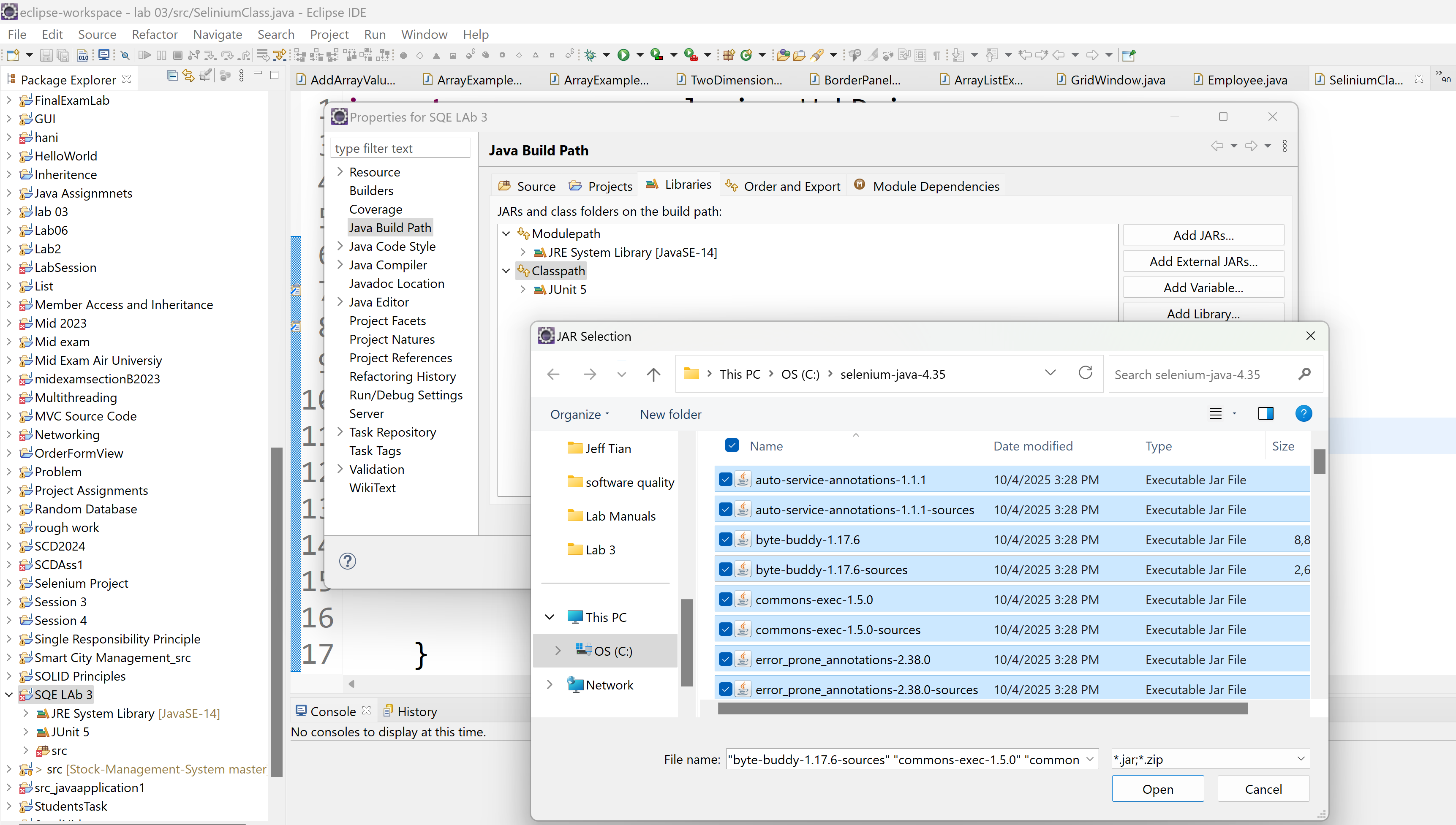
1. Go to: <https://www.selenium.dev/>
2. Click on download in the menu bar.
3. Click on the java version: Stable: [4.36.0 (October 02, 2025)](https://github.com/SeleniumHQ/selenium/releases/download/selenium-4.36.0/selenium-java-4.36.0.zip) as shown in the figure



1. When you click on java, then the zip file of selenium-java-4.36.0 will be downloaded.
2. Extract selenium-java-4.36.0 zip file.
3. Open Eclipse
4. Create a new project with the name of **Automation**.
5. Right click on the **Automation** project in the package explorer, then select Build Path and then configure build path.

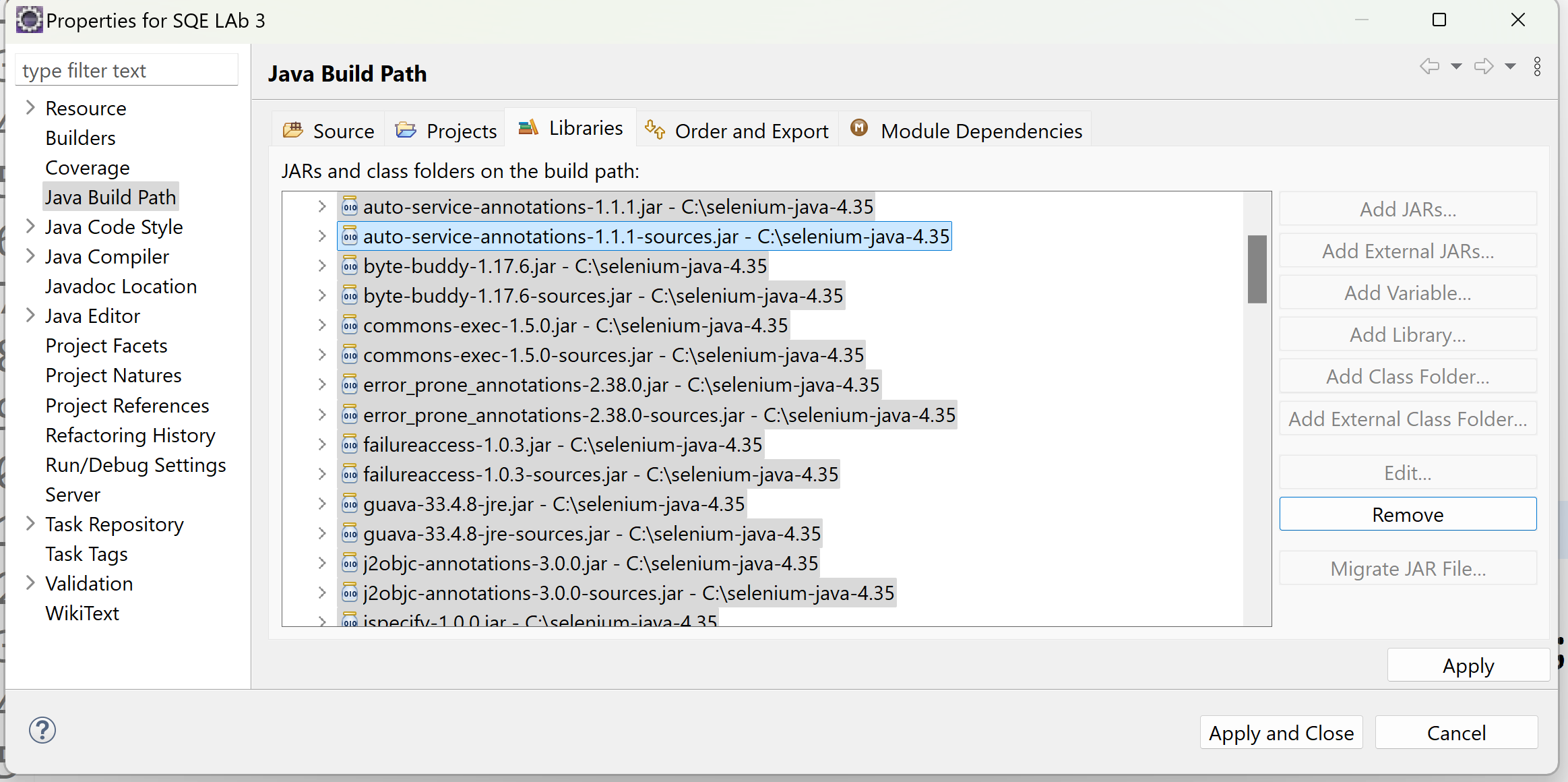




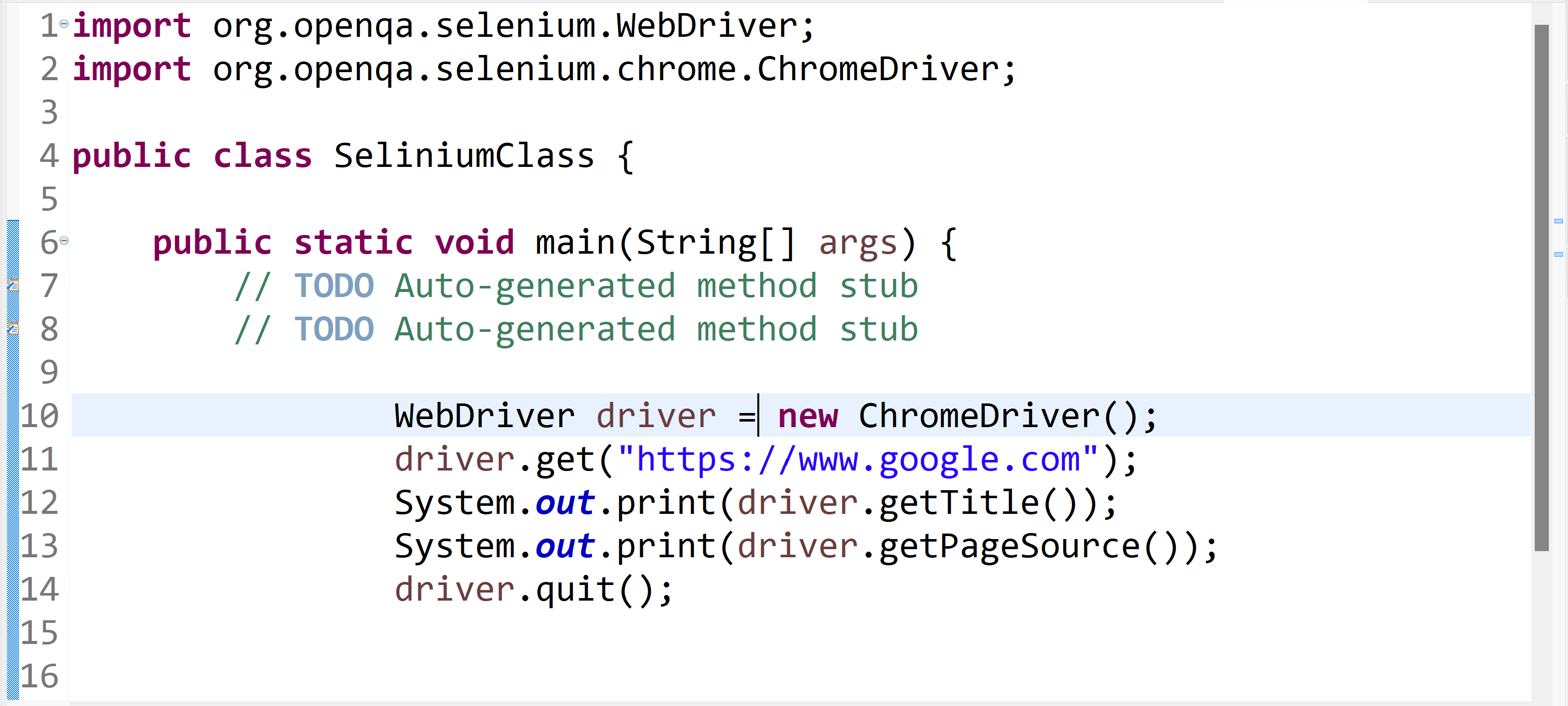




After you select all files and open, then click on the “apply and close”.



1. Once you are done with adding the jar files, then create a Java class with the name of **SeleniumTest** class containing the main method
2. Write the following code in eclipse IDE as follows:



## Introduction to WebDriver

## WebDriver and API of WebDriver

If you want to interaction with chrome browser then you can instantiate chrome driver using WebDriver.

## What is Selenium WebDriver?

Selenium WebDriver is a tool used to automate web browsers. It directly communicates with the browser using that browser’s built-in automation support (often called its native API). The exact way WebDriver interacts with a browser depends on which browser you’re using.

For example:

* **Chrome uses ChromeDriver**
* **Firefox uses GeckoDriver**
* **Safari uses SafariDriver  
  …and so on.**

This allows Selenium to open browsers, click buttons, fill forms, navigate pages, and test web apps just like a human would.

## Types of WebDrivers Available in Selenium

The image lists several driver implementations that can be used with Selenium:

* **ChromeDriver** → For automating Google Chrome
* **EventFiringWebDriver** → A wrapper that lets you attach event listeners (e.g., logging, debugging)
* **FirefoxDriver** → For automating Mozilla Firefox
* **HtmlUnitDriver** → Headless browser driver (no UI), mainly for faster tests
* **InternetExplorerDriver** → For automating Internet Explorer (older Microsoft browser)
* **PhantomJSDriver** → Headless browser (deprecated now)
* **RemoteWebDriver** → For running tests on remote machines or Selenium Grid
* **SafariDriver** → For automating Safari on macOS

## Purpose in Automation

These drivers give Selenium the ability to:

* Launch browsers
* Navigate to pages
* Interact with elements (click, type, scroll)
* Run automated tests
* Retrieve results

The features available depend on the browser and its driver.

## Tasks

1. Instantiate the objects of WebDriver for all browsers using the above setup of Java Selenium Project. These browsers can be such as Google Chrome, Internet Explorer, Safari, Mozilla FireFox, Microsoft Edge, Opera, etc.
2. Apply all relevant the top-level WebDriver methods that let you control the browser itself without elements. This can be done only for one browser for your own choice most probably the Google Chrome
3. Demonstrate to your Instructor.

**References:**

[1] Selenium, Chrome specific functionality, [Available online] https://www.selenium.dev/documentation/webdriver/browsers/chrome/.[Accessed 15 Oct 2025]

[2] Selenium, Edge specific functionality, [Available online] https://www.selenium.dev/documentation/webdriver/browsers/edge/.[Accessed 15 Oct 2025]

[3] Selenium, Firefox specific functionality, [Available online] https://www.selenium.dev/documentation/webdriver/browsers/firefox/.[Accessed 15 Oct 2025]

[4] Selenium, IE specific functionality, [Available online] https://www.selenium.dev/documentation/webdriver/browsers/internet\_explorer/.[Accessed 15 Oct 2025].

[5] Selenium, Safari specific functionality, [Available online] https://www.selenium.dev/documentation/webdriver/browsers/safari/.[Accessed 15 Oct 2025].

[6] Selenium, [Available online] https://www.selenium.dev/