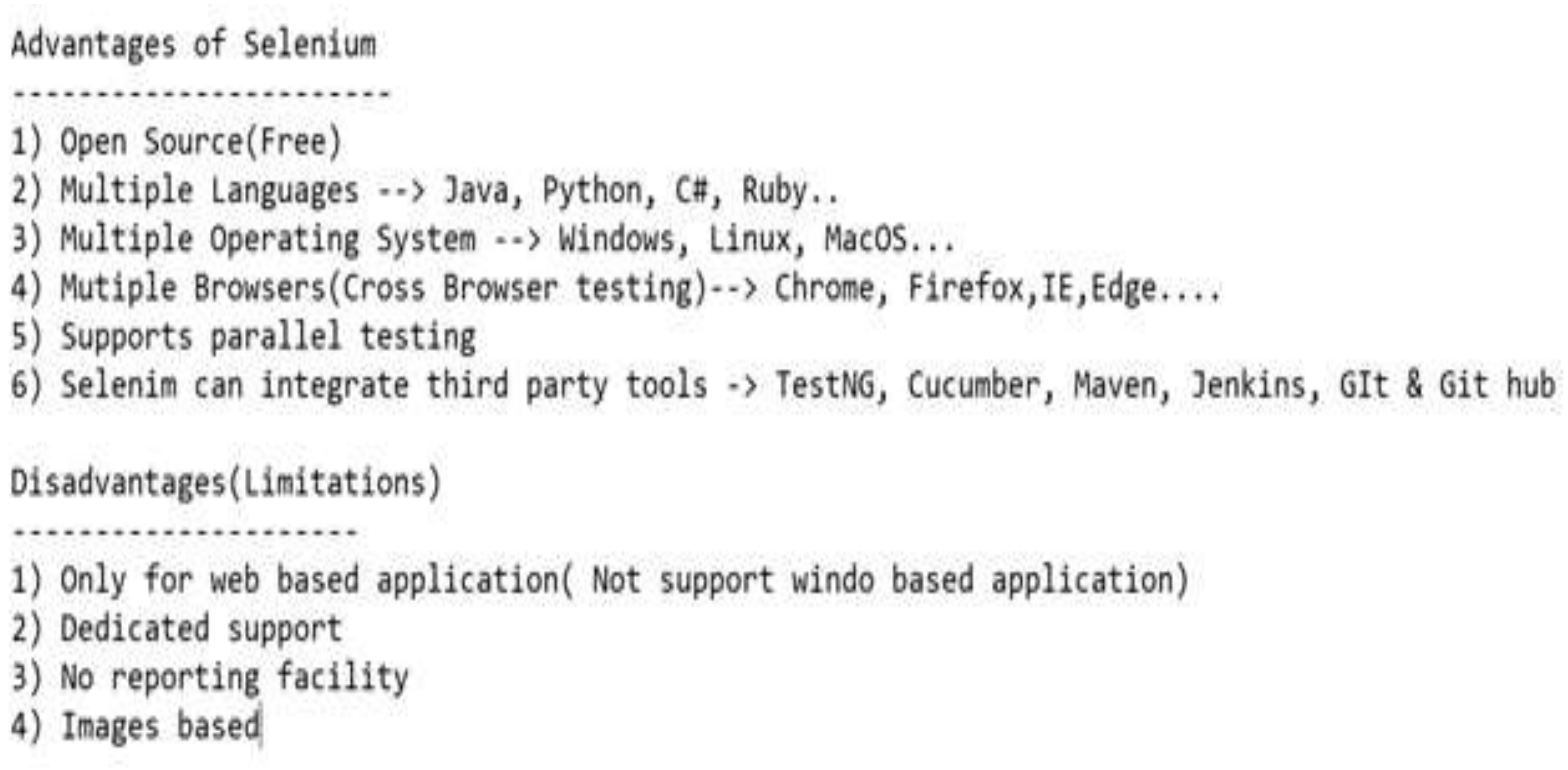
AUTOMATION TESTING

```
Functional testing tools
------------------------
QTP/UFT
Selenium

Performance testing
----------
Jmeter
LoadRunner

test Mangement tools
--------
Ex: Quality Center, MS VSTP
Test cases
Defects reporting
Execution status
report generates

Defect reporting
-------------
BugZilla
Remedy
Devrack

Jira--> test management, defect management, Dashboads
```

- Selenium is Functional and Regression Testing Tool



```
Advantages of Selenium
-----------------------
1) Open Source(Free)
2) Multiple Languages --> Java, Python, C#, Ruby..
3) Multiple Operating System --> Windows, Linux, MacOS...
4) Mutiple Browsers(Cross Browser testing)--> Chrome, Firefox,IE,Edge....
5) Supports parallel testing
6) Selenim can integrate third party tools -> TestNG, Cucumber, Maven, Jenkins, GIt & Git hub

Disadvantages(Limitations)
---------------------
1) Only for web based application( Not support windo based application)
2) Dedicated support
3) No reporting facility
4) Images based
```

Disadvantages(Limitations)
----
1) Only for web based application(Not support window based application) ---> AutoIT,sikuli
2) Dedicated support   --> Forums available
3) No reporting facility   --> TestNG, Extents
4) Images based --> Certain level

Selenium --> Selenium is set of componets/tools.
Selenium is a suite whihc contains multiple componets/tools

Selenium Suite
----
1) Selenium IDE
2) Selenium RC
3) Selenium WebDriver
4) Selenium grid

Selenium 1---> IDE, RC, Grid
Selenium 2--> IDE,WebDriver, GRID
Selenium 3 --> WebDriver, grid

IDE --> plug-in for firefox browser, record and play back tool
RC--> RC server will interact with browser and execute Automating test cases (in direct)
WebDriver ---> WebDriver ia an API which will automate your test cases.
Grid--> upu can execute test cases remotely.

Components of Selenium

The components of Selenium are :-

- Selenium IDE.
- Selenium RC.
- Selenium Webdriver.
- Selenium Grid.

Selenium IDE:-

- Selenium Integrated Development Environment is an important part of the Selenium suite. It was first developed as a Firefox plugin, however now it is available in both Firefox and Chrome browser.

Some of the features of Selenium IDE:-

- The recording, debugging and editing of the functional tests can be done in Selenium IDE.
- The scripts in Selenium IDE are developed in Selenese which is a scripting language in Selenium.
- Selenium commands help us to perform tasks like clicking a button or link, taking input in an edit box, obtaining a text from a web element and so on.

Selenium RC

- Selenium Remote Control is a server implemented in Java. It can accept commands for browsers using the HTTP.

Some of the features of Selenium RC

- Automation tests in Selenium RC can be developed in any programming languages like Java, Python, C# and so on.
- To initiate test execution, we have to create an instance of the Selenium RC server.

Selenium WebDriver

- Selenium webdriver was developed after Selenium RC. It receives commands and passes them to the browser. This is done with the help of the browser drivers that sends commands to the browser and obtains the results.

Some of the features of Selenium webdriver −

- Automation tests can be written in multiple programming languages like the Java, C#, Python, JavaScript, and so on.
- Selenium webdriver supports browsers like Chrome, Firefox, Safari, IE, and so on.
- Selenium webdriver works on more than one platforms like Windows, Mac, Linux, Android, and so on.
- Selenium webdriver does not require a server to initiate test execution and it communicates directly with the browser.
- Selenium webdriver is open-source and comes without any licensing cost.

Selenium Grid:

- Selenium Grid is mainly used for parallel testing. It enables us to execute varied tests in multiple machines simultaneously.

Some of the features of Selenium Grid:

- Presence of a hub machine which directs the execution on multiple machines.
- Selenium Grid supports testing on multiple browsers and platforms.
- Reduces execution time by allowing parallel execution of tests.

Selenium WebDriver

- Selenium WebDriver is the most important component of Selenium Tool's Suite. The latest release "Selenium 2.0" is integrated with WebDriver API which provides a simpler and more concise programming interface.
- Selenium WebDriver was first introduced as a part of Selenium v2.0. The initial version of Selenium i.e Selenium v1 consisted of only IDE, RC and Grid. However, with the release of Selenium v3, RC has been deprecated and moved to legacy package.
- In WebDriver, test scripts can be developed using any of the supported programming languages and can be run directly in most modern web browsers. Languages supported by WebDriver include C#, Java, Perl, PHP, Python and Ruby.

- Before learning the concepts of Selenium WebDriver, we should be well versed with any of the supported programming languages. Currently, Selenium Web driver is most popular with Java and C# and we are using Selenium with java.
- Selenium WebDriver performs much faster as compared to Selenium RC because it makes direct calls to the web browsers. RC on the other hand needs an RC server to interact with the browser.
- WebDriver has a built-in implementation of Firefox driver (Gecko Driver). For other browsers, you need to plug-in their browser specific drivers to communicate and run the test.

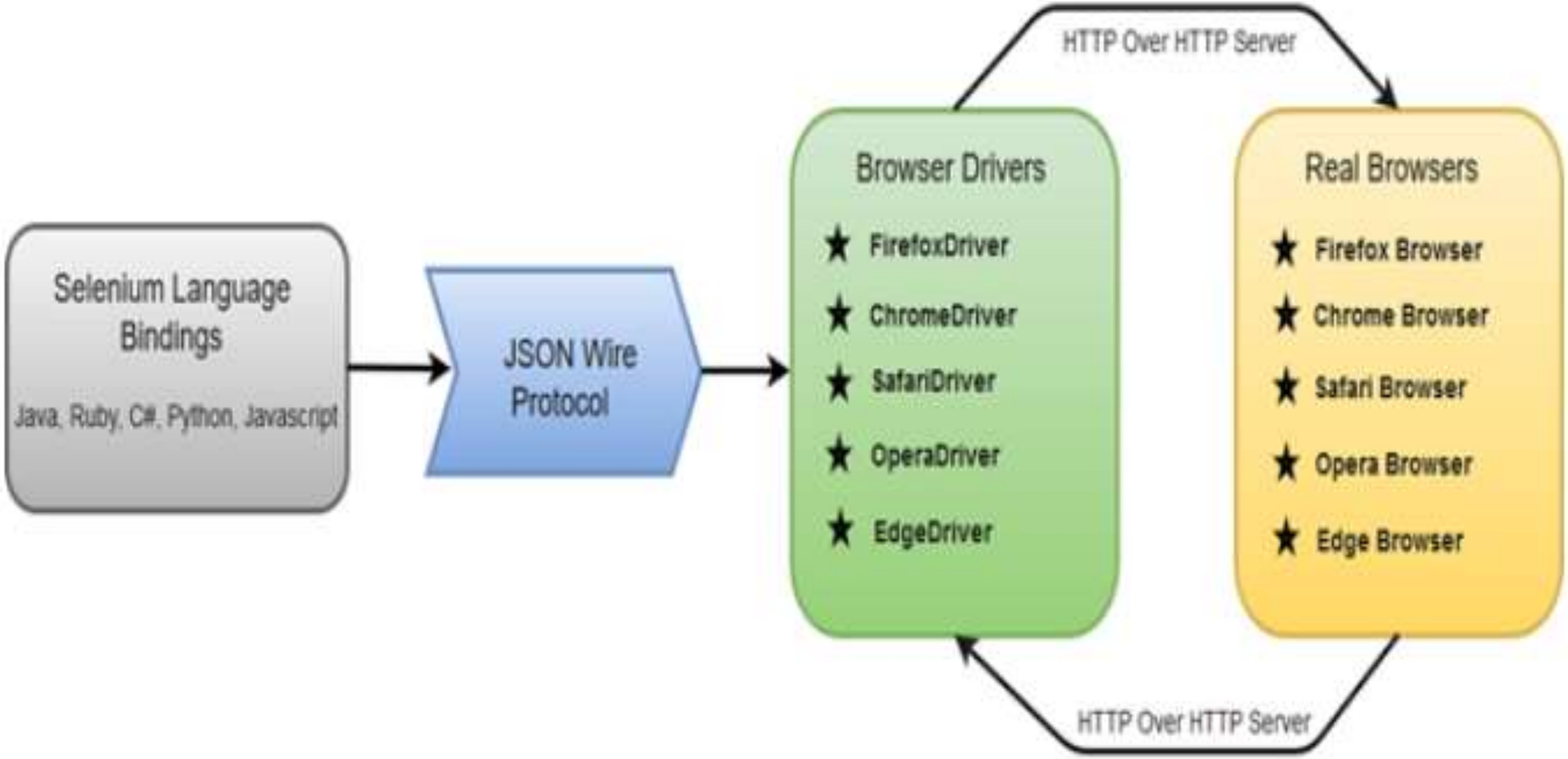
Most commonly used WebDriver's include:

- Google Chrome Driver
- Internet Explorer Driver
- Opera Driver
- Microsoft Edge Driver

Selenium WebDriver- Architecture:

- Selenium WebDriver API provides communication facility between languages and browsers.

The following image shows the architectural representation of Selenium WebDriver:-


Selenium Language Bindings
Java, Ruby, C#, Python, Javascript
JSON Wire Protocol
Browser Drivers
FirefoxDriver
ChromeDriver
SafariDriver
OperaDriver
EdgeDriver
HTTP Over HTTP Server
Real Browsers
Firefox Browser
Chrome Browser
Safari Browser
Opera Browser
Edge Browser
HTTP Over HTTP Server

There are four basic components of WebDriver Architecture:

- Selenium Language Bindings
- JSON Wire Protocol
- Browser Drivers
- Real Browsers

- Selenium Language Bindings

Selenium developers have built language bindings/Selenium Client Libraries in order to support multiple languages. For instance, if we want to use the browser driver in java, use the java bindings. All the supported language bindings can be downloaded from the official website of Selenium.

JSON Wire Protocol

- JSON (JavaScript Object Notation) is an open standard for exchanging data on web. It supports data structures like object and array. So, it is easy to write and read data from JSON.
- JSON Wire Protocol provides a transport mechanism to transfer data between a server and a client.

Browser Drivers

- Selenium uses drivers, specific to each browser in order to establish a secure connection with the browser without revealing the internal logic of browser's functionality. The browser driver is also specific to the language used for automation such as Java, C#, etc.
- When we execute a test script using WebDriver, the following operations are performed internally:-

- HTTP request is generated and sent to the browser driver for each Selenium command.
- The driver receives the HTTP request through HTTP server.
- HTTP Server decides all the steps to perform instructions which are executed on browser.
- Execution status is sent back to HTTP Server which is subsequently sent back to automation script.

Browsers

Browsers supported by Selenium WebDriver:

- Internet Explorer
- Mozilla Firefox
- Google Chrome
- Microsoft Edge etc.

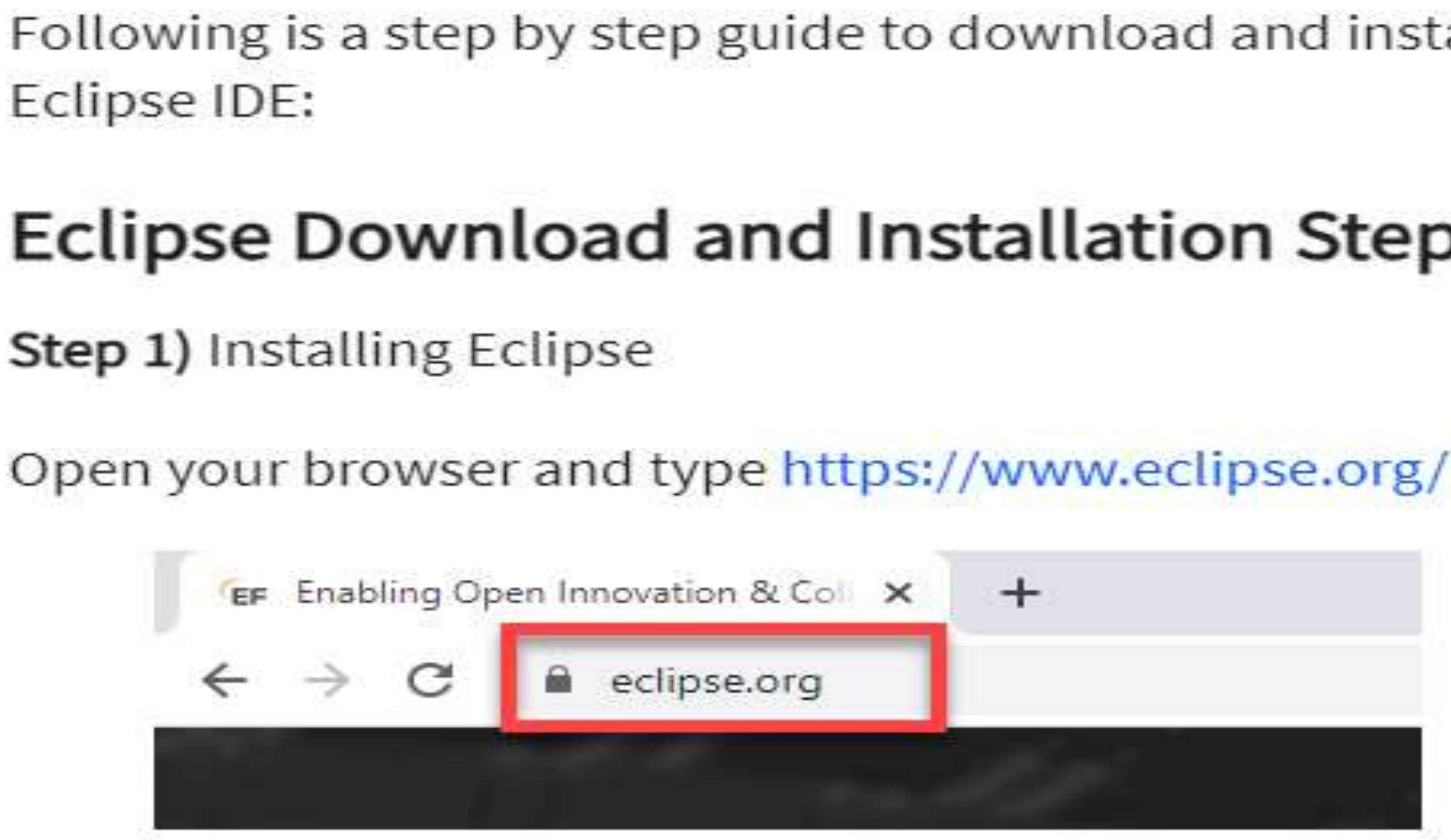
Selenium WebDriver- Features

Some of the most important features of Selenium WebDriver are:

- Multiple Browser Support
- Multiple Languages Support
- Speed
- Simple Commands

Installation process-Eclipse

https://www.guru99.com/install-eclipse-java.html



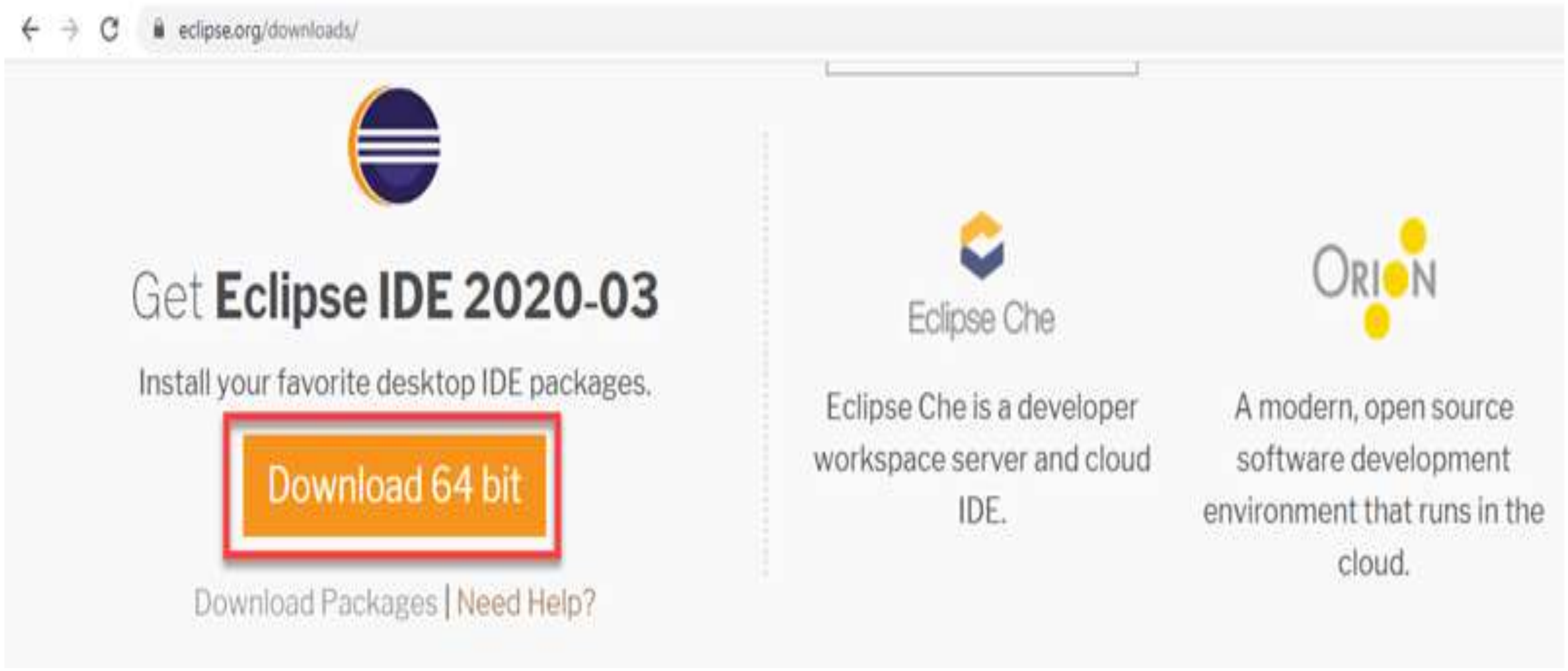
Following is a step by step guide to download and install Eclipse IDE:

Eclipse Download and Installation Steps

Step 1) Installing Eclipse

Open your browser and type https://www.eclipse.org/

- Step-2): Click on "Download" Button.
- Step 3): Click on "Download 64 bit" button.


eclipse.org/downloads/

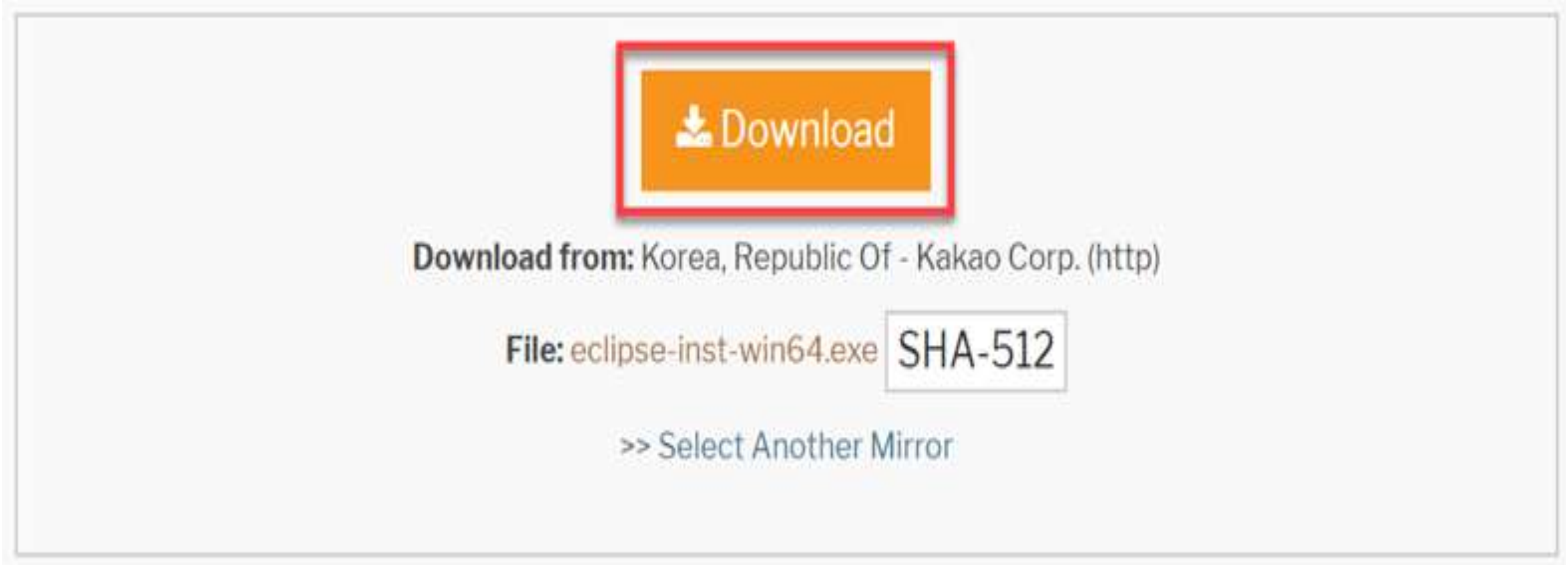
Get **Eclipse IDE 2020-03**

Install your favorite desktop IDE packages.

Eclipse Che is a developer workspace server and cloud IDE.

A modern, open source software development environment that runs in the cloud.

Step 4) click on “Download” Button.



Download from: Korea, Republic Of - Kakao Corp. (http)

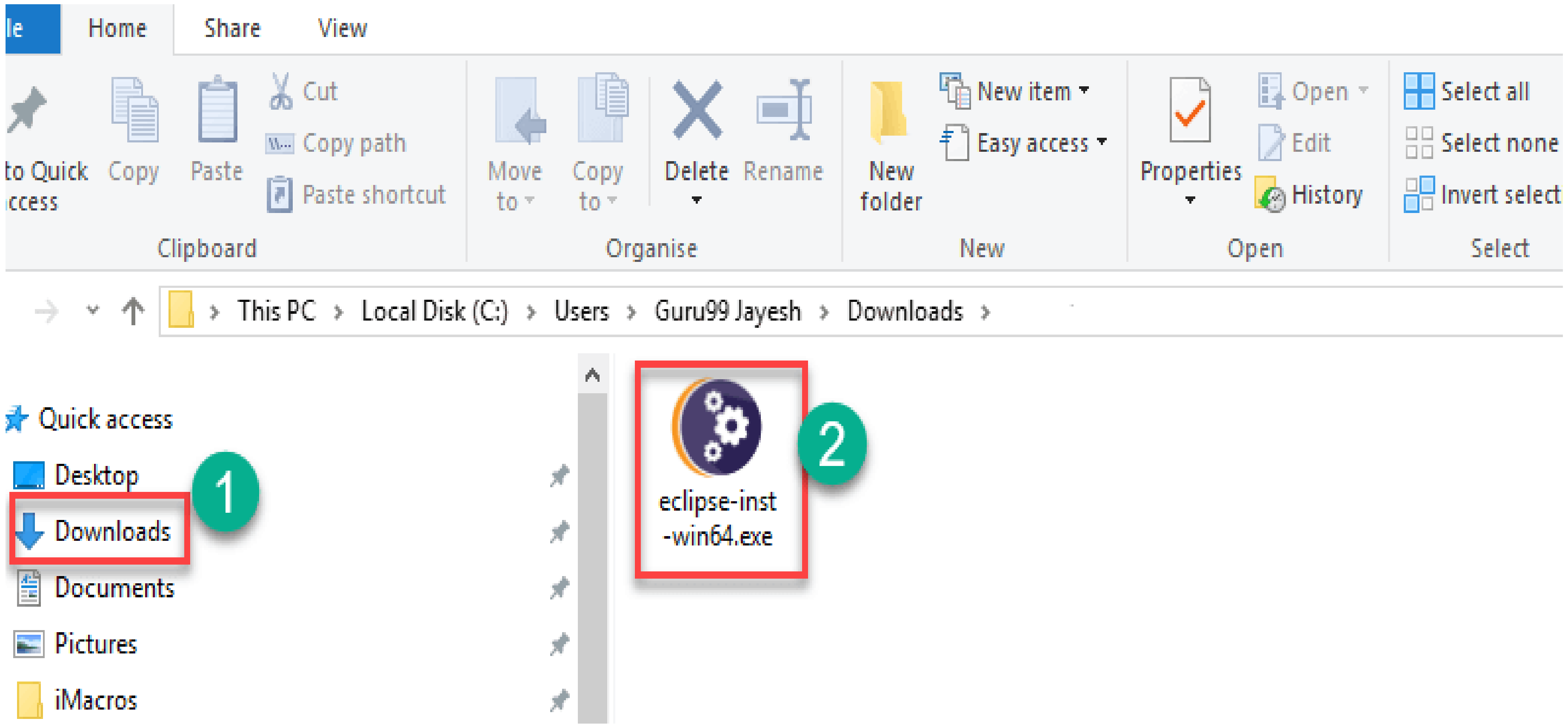
File: eclipse-inst-win64.exe SHA-512

>> Select Another Mirror

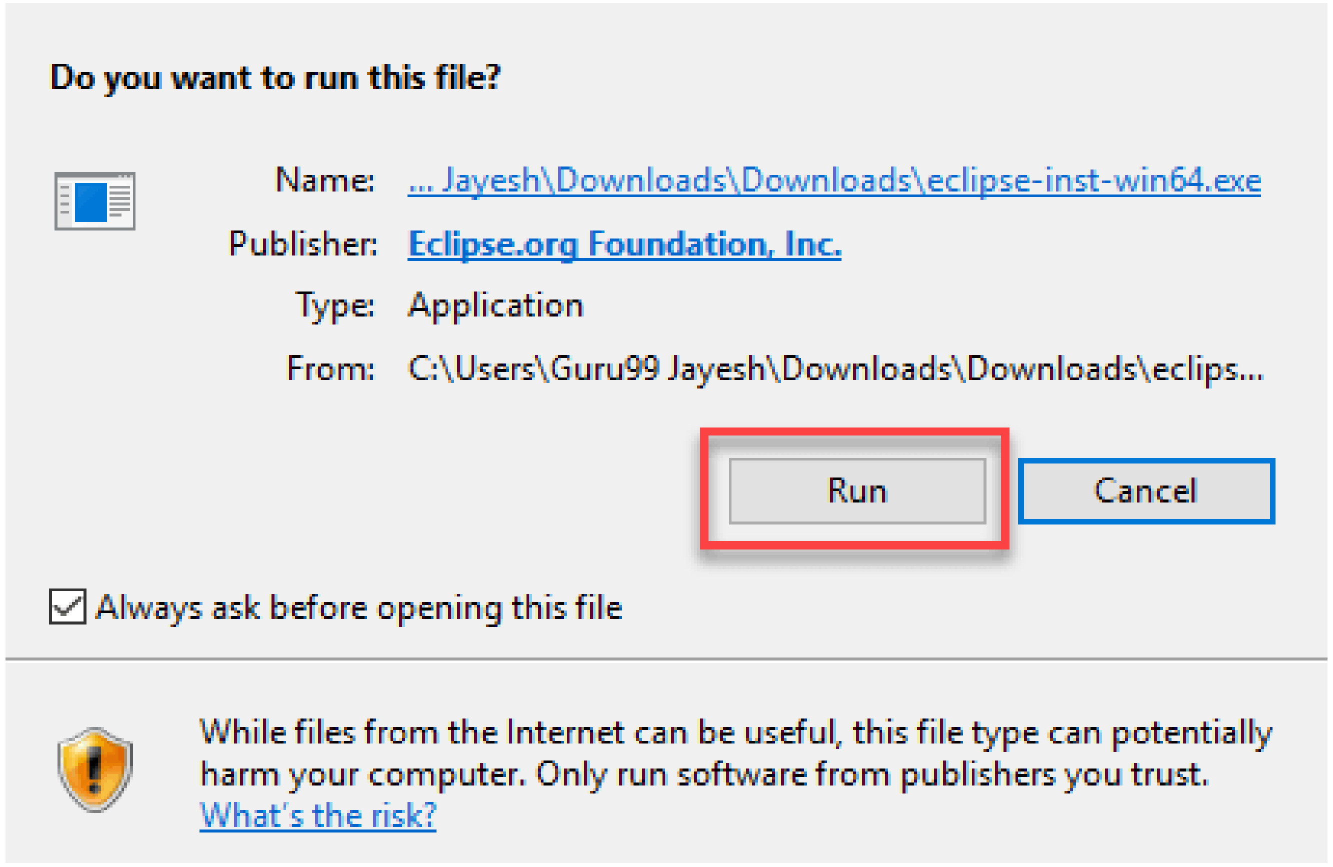
Install Eclipse.

1.Click on “downloads” in Windows file explorer.

2.Click on “eclipse-inst-win64.exe” file.



Step 5) Click on Run button



Do you want to run this file?

Name: ... Jayesh\Downloads\Downloads\eclipse-inst-win64.exe

Publisher: Eclipse.org Foundation, Inc.

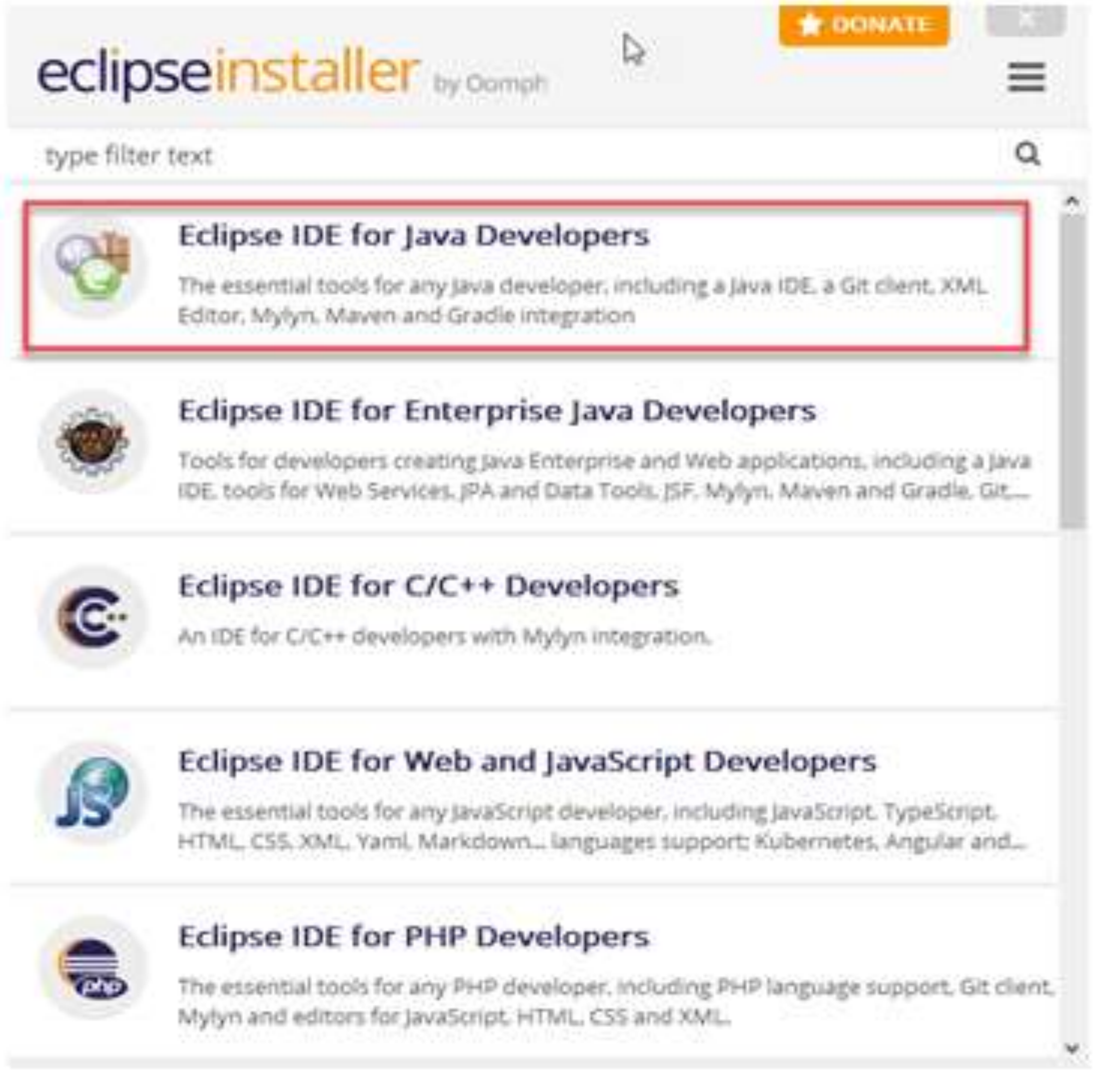
Type: Application

From: C:\Users\Guru99 Jayesh\Downloads\Downloads\eclips...

☑ Always ask before opening this file

While files from the Internet can be useful, this file type can potentially harm your computer. Only run software from publishers you trust. What's the risk?

Step 6) Click on "Eclipse IDE for Java Developers"



Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

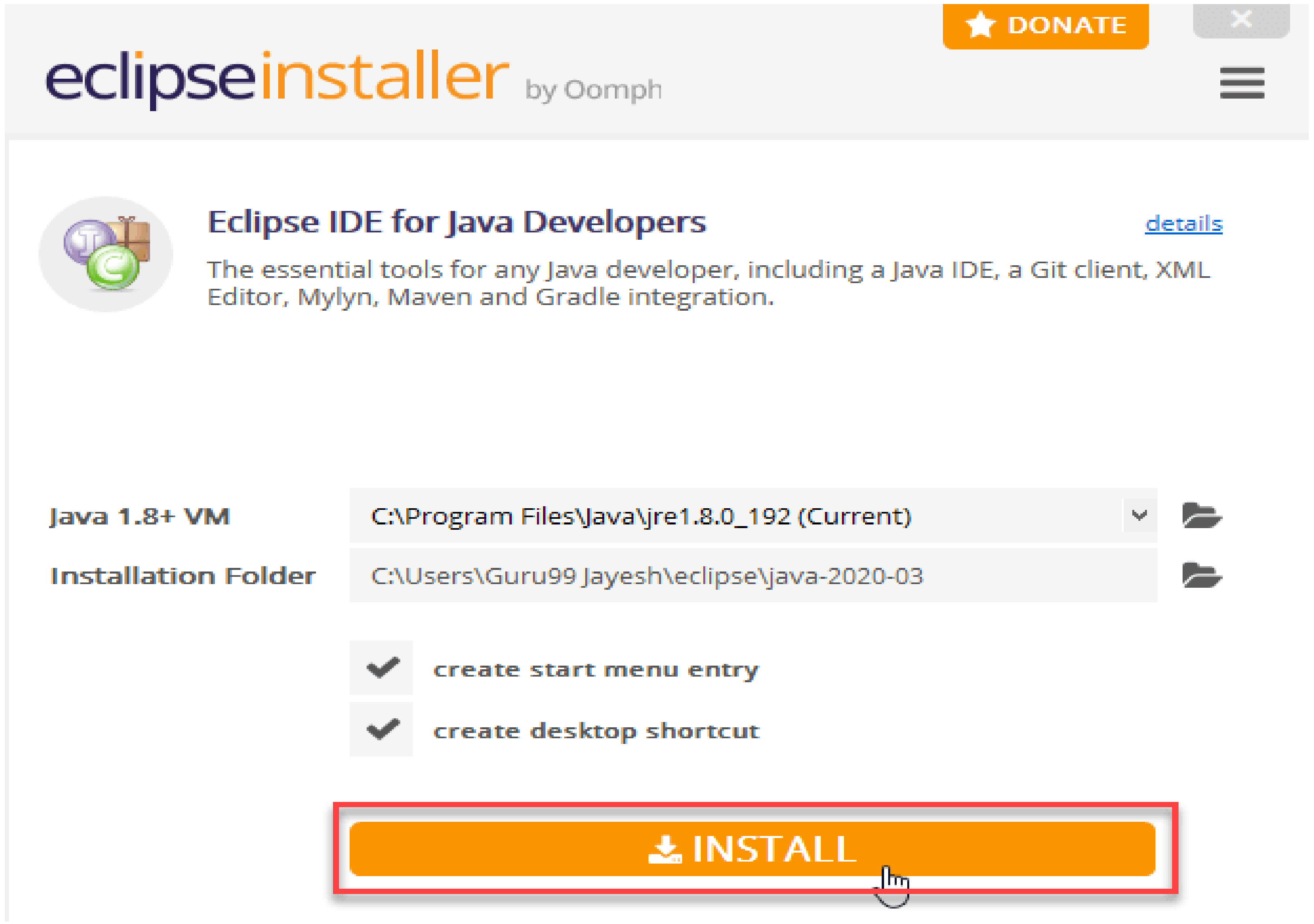
Eclipse IDE for Enterprise Java Developers
Tools for developers creating Java Enterprise and Web applications, including a Java IDE, tools for Web Services, JPA and Data Tools, JSF, Mylyn, Maven and Gradle, Git,...

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.

Eclipse IDE for Web and JavaScript Developers
The essential tools for any JavaScript developer, including JavaScript, TypeScript, HTML, CSS, XML, Yaml, Markdown... languages support; Kubernetes, Angular and...

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.

Step 7) Click on “INSTALL” button



Eclipse IDE for Java Developers

details

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

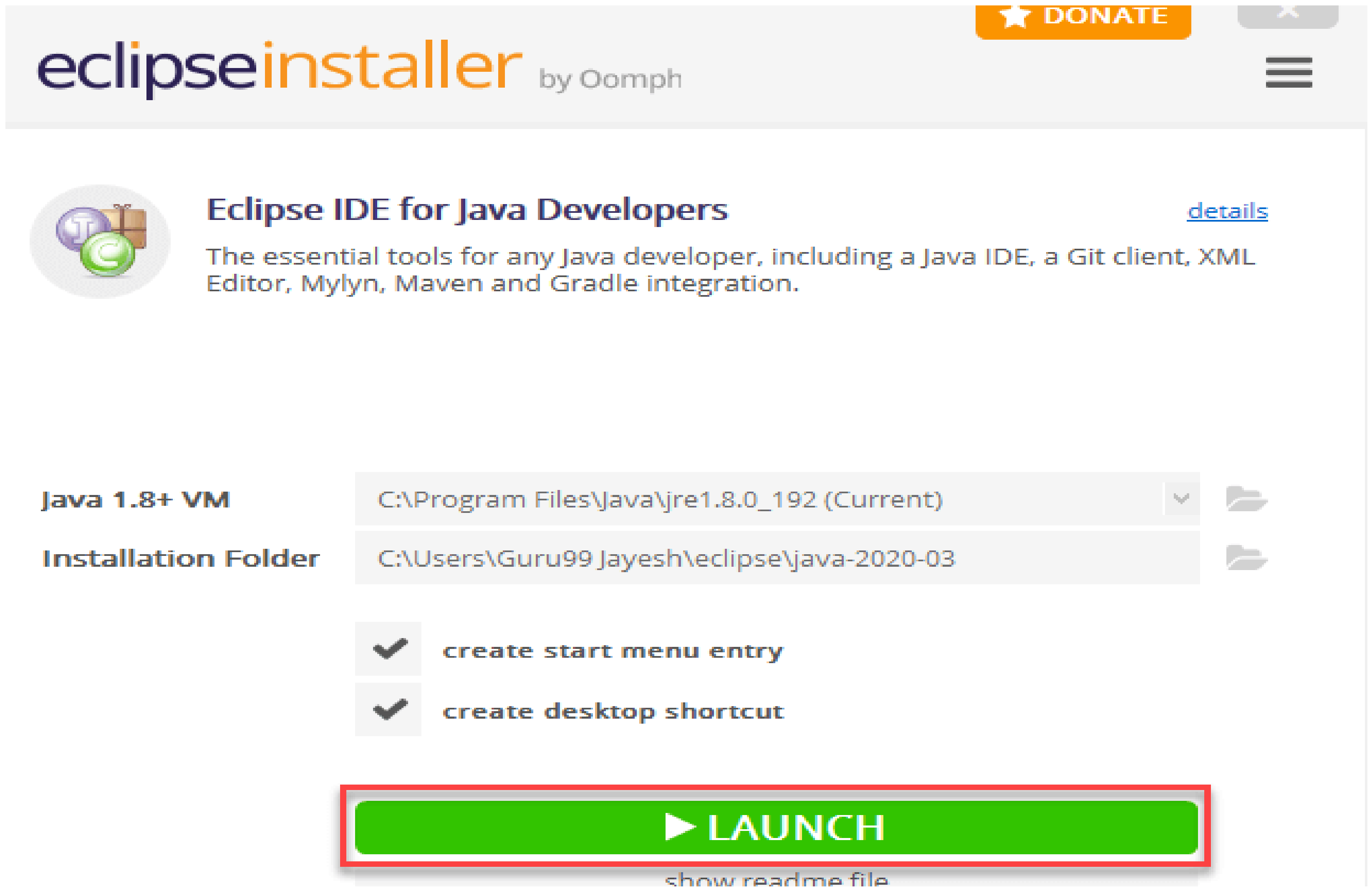
Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_192 (Current)

Installation Folder: C:\Users\Guru99 Jayesh\eclipse\java-2020-03

✓ create start menu entry

✓ create desktop shortcut

Step 8) Click on “LAUNCH” button.



Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

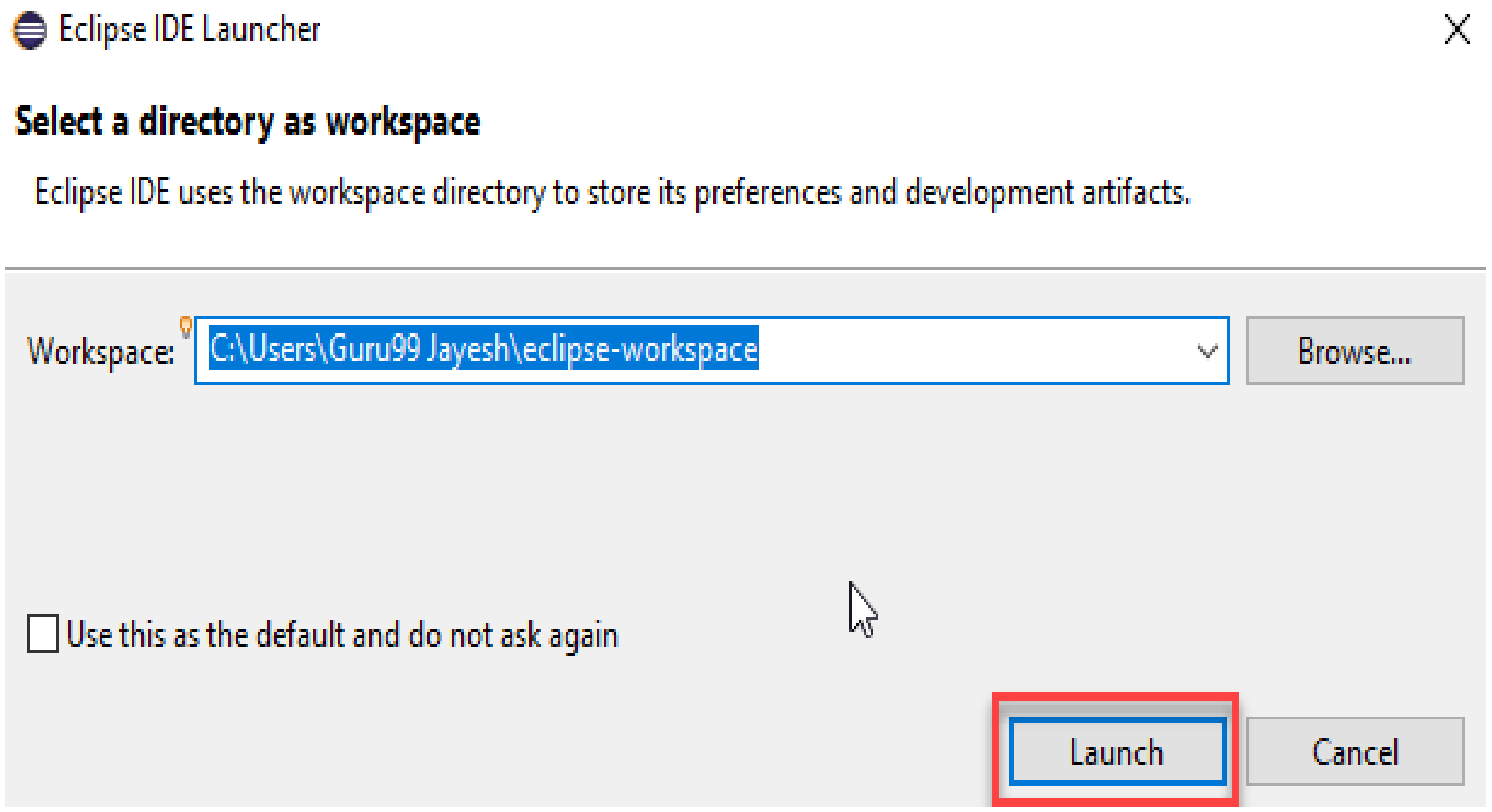
Java 1.8+ VM: C:\Program Files\Java\jre1.8.0_192 (Current)

Installation Folder: C:\Users\Guru99 Jayesh\eclipse\java-2020-03

✓ create start menu entry

✓ create desktop shortcut

Step 9) Click on "Launch" button



Eclipse IDE Launcher

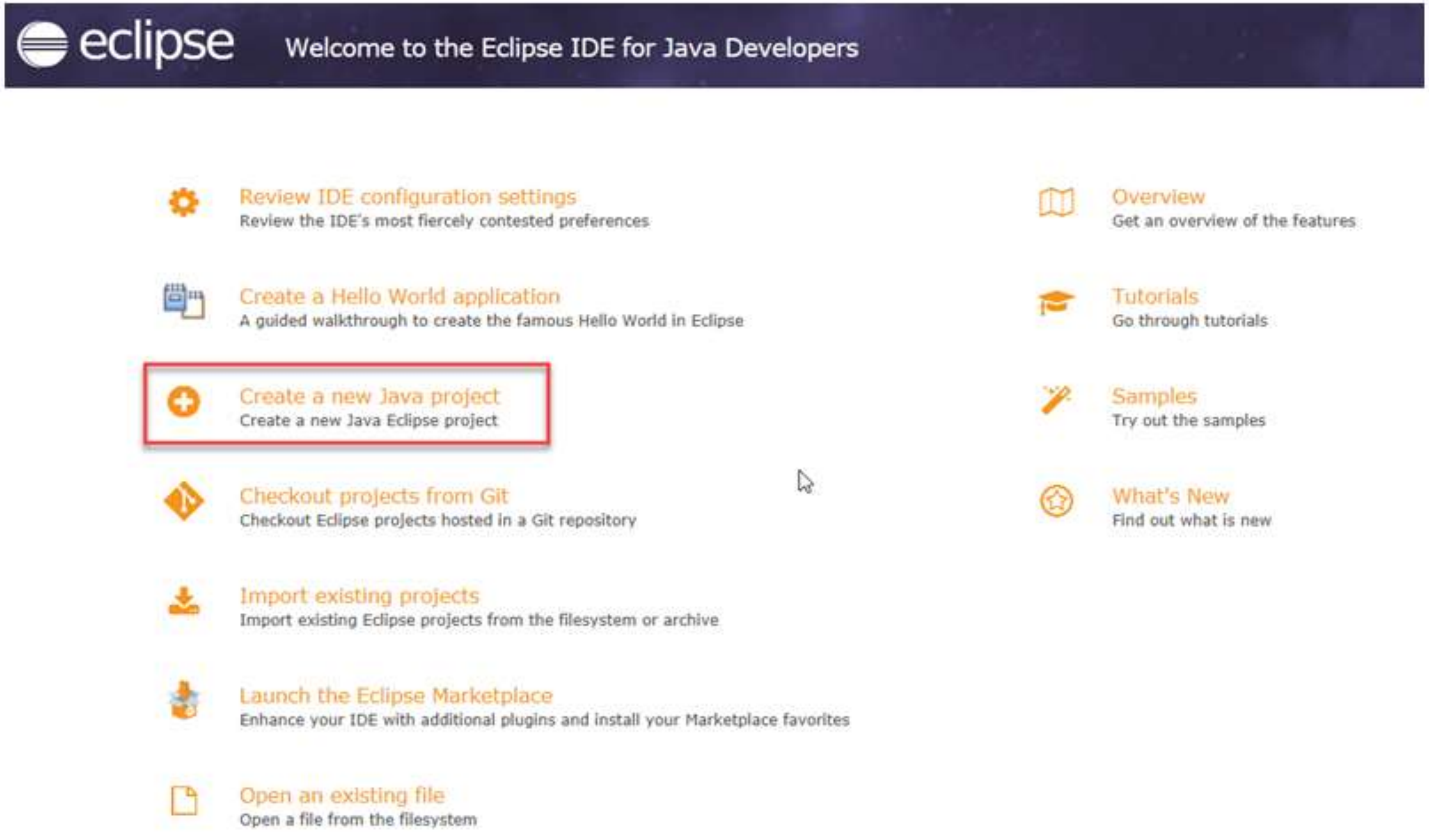
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\Guru99 Jayesh\eclipse-workspace

☐ Use this as the default and do not ask again

Step 10) Click on “Create a new Java project” link



eclipse Welcome to the Eclipse IDE for Java Developers

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse

Create a new Java project
Create a new Java Eclipse project

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins and install your Marketplace favorites

Open an existing file
Open a file from the filesystem

Overview
Get an overview of the features

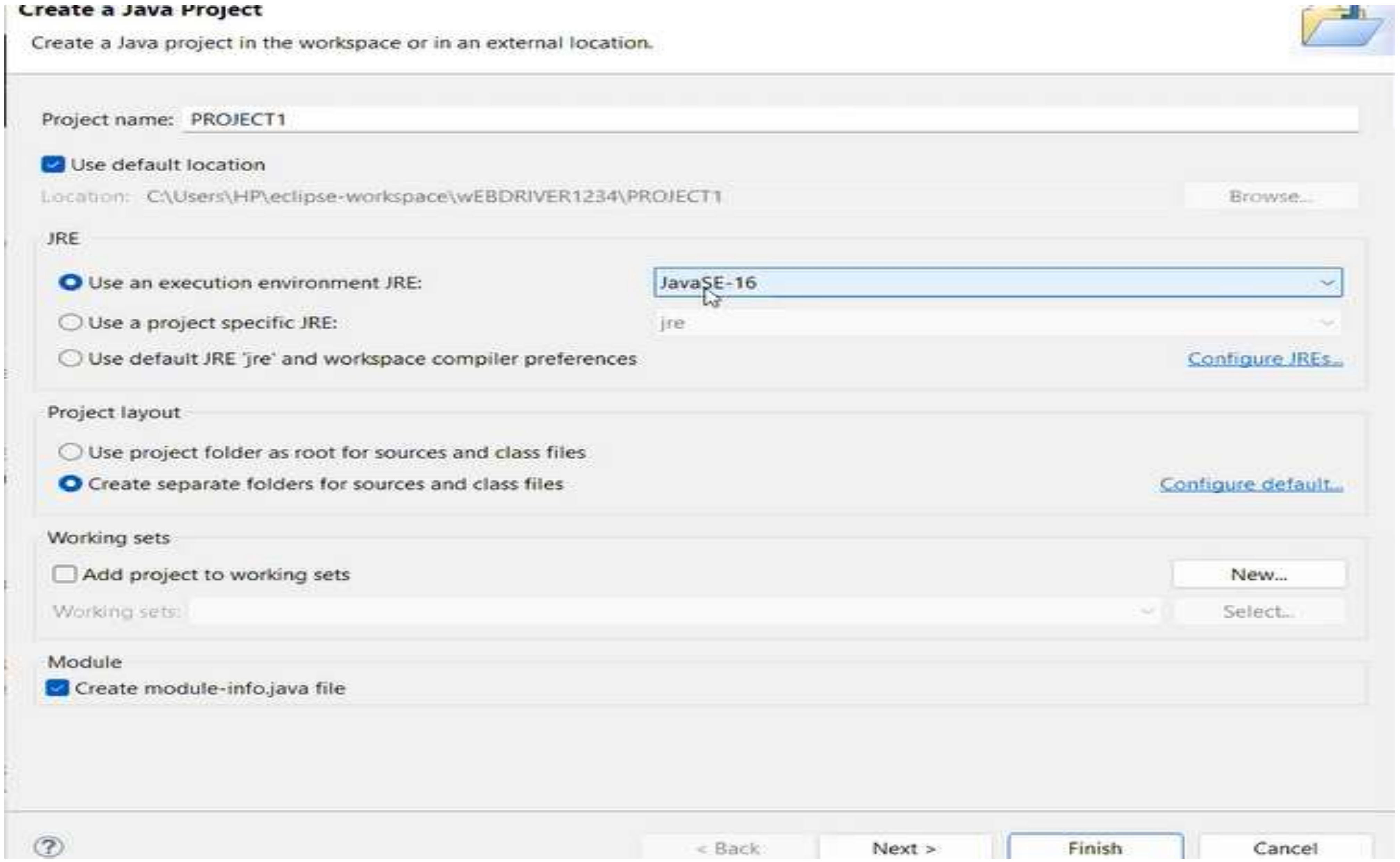
Tutorials
Go through tutorials

Samples
Try out the samples

What's New
Find out what is new

STEPS FOR CREATING PROJECT IN ECLIPSE

- GOTO NEW
- SELECT JAVA PROJECT
- SPECIFY PROJECT NAME
- CLICK NEXT
- CLICK ON DON’T CREATE BUTTON FOR (crete module in java info)
- Click next
- Click finish



Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: PROJECT1

[✓] Use default location

Location: C:\Users\HP\eclipse-workspace\wEBDRIVER1234\PROJECT1

JRE

[✓ Use an execution environment JRE:] JavaSE-16

Use a project specific JRE: jre

Use default JRE 'jre' and workspace compiler preferences

Project layout

Use project folder as root for sources and class files

[✓ Create separate folders for sources and class files]

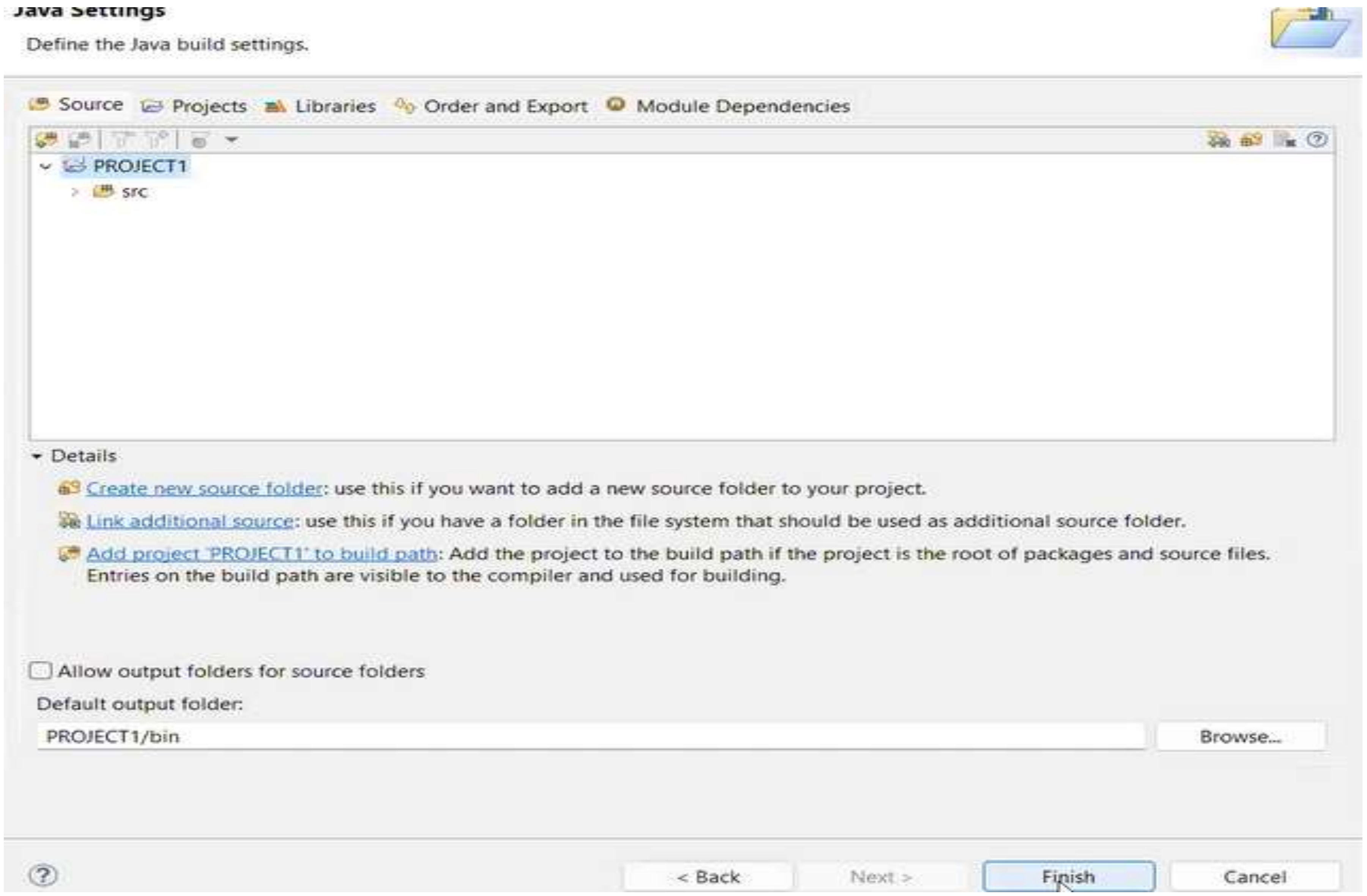
Working sets

[ ] Add project to working sets

Working sets:

Module

[✓] Create module-info.java file


Java Settings

Define the Java build settings.

PROJECT1
src

Details

Create new source folder: use this if you want to add a new source folder to your project.

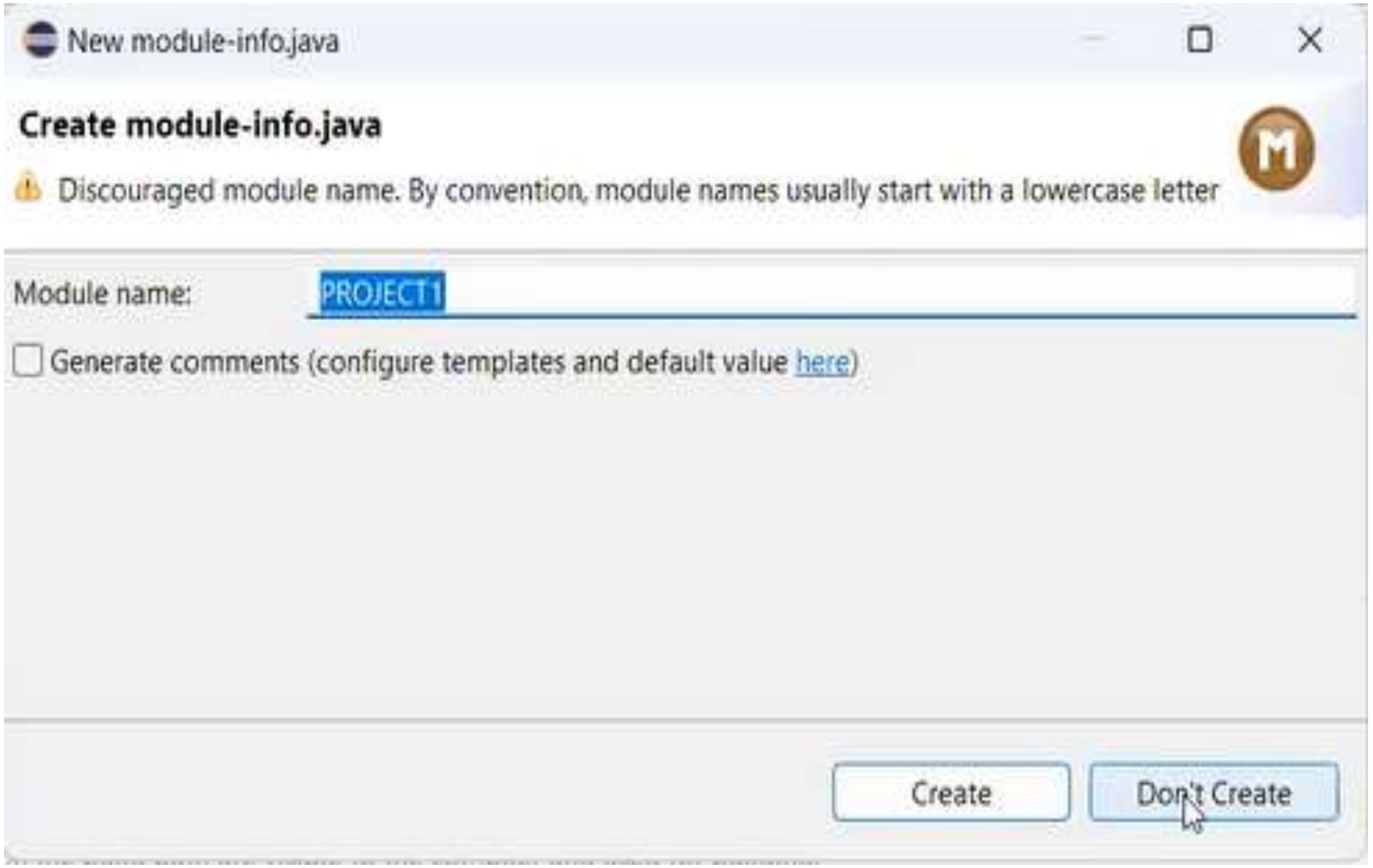
Link additional source: use this if you have a folder in the file system that should be used as additional source folder.

Add project 'PROJECT1' to build path: Add the project to the build path if the project is the root of packages and source files. Entries on the build path are visible to the compiler and used for building.

☐ Allow output folders for source folders

Default output folder:

PROJECT1/bin



Create module-info.java

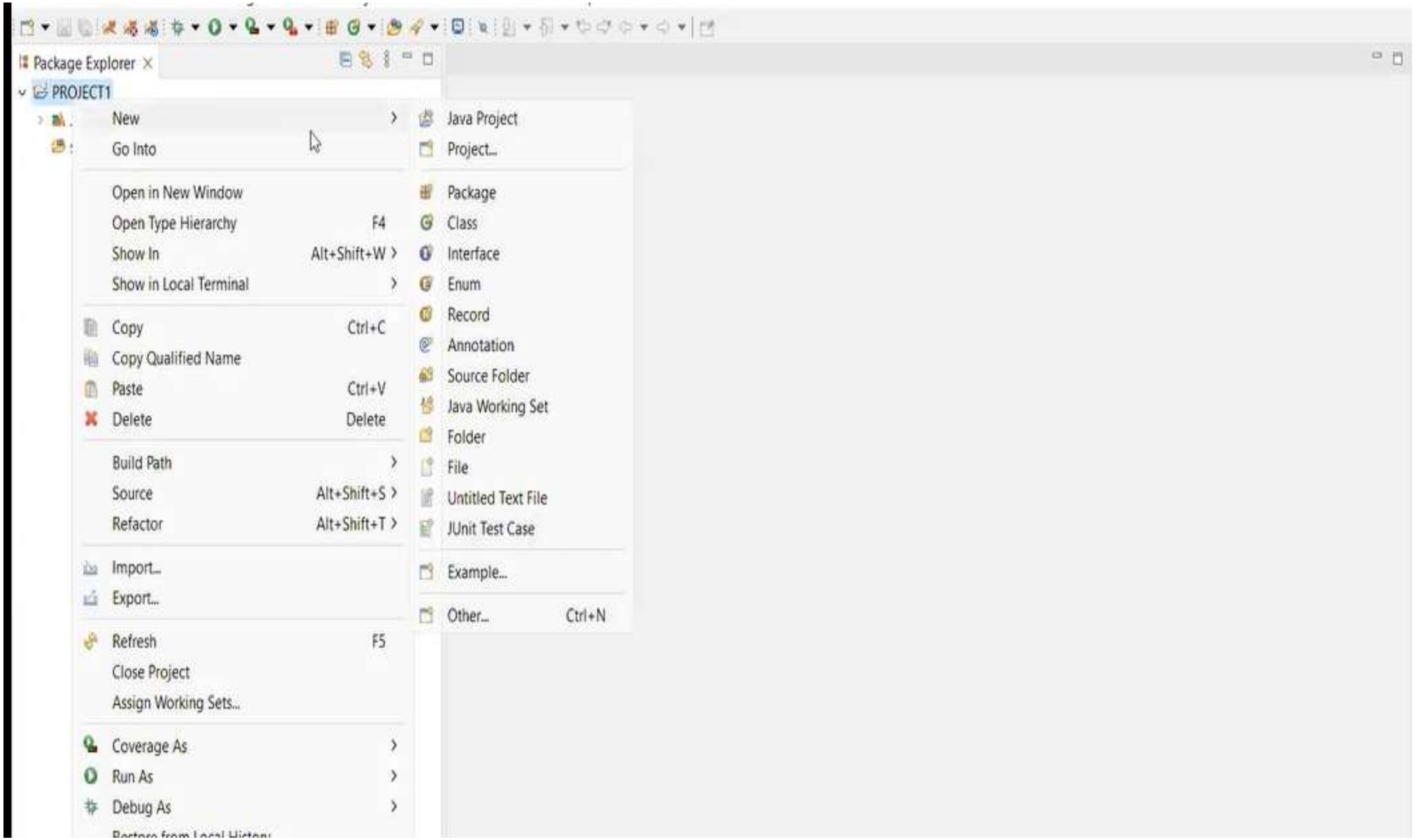
Discouraged module name. By convention, module names usually start with a lowercase letter

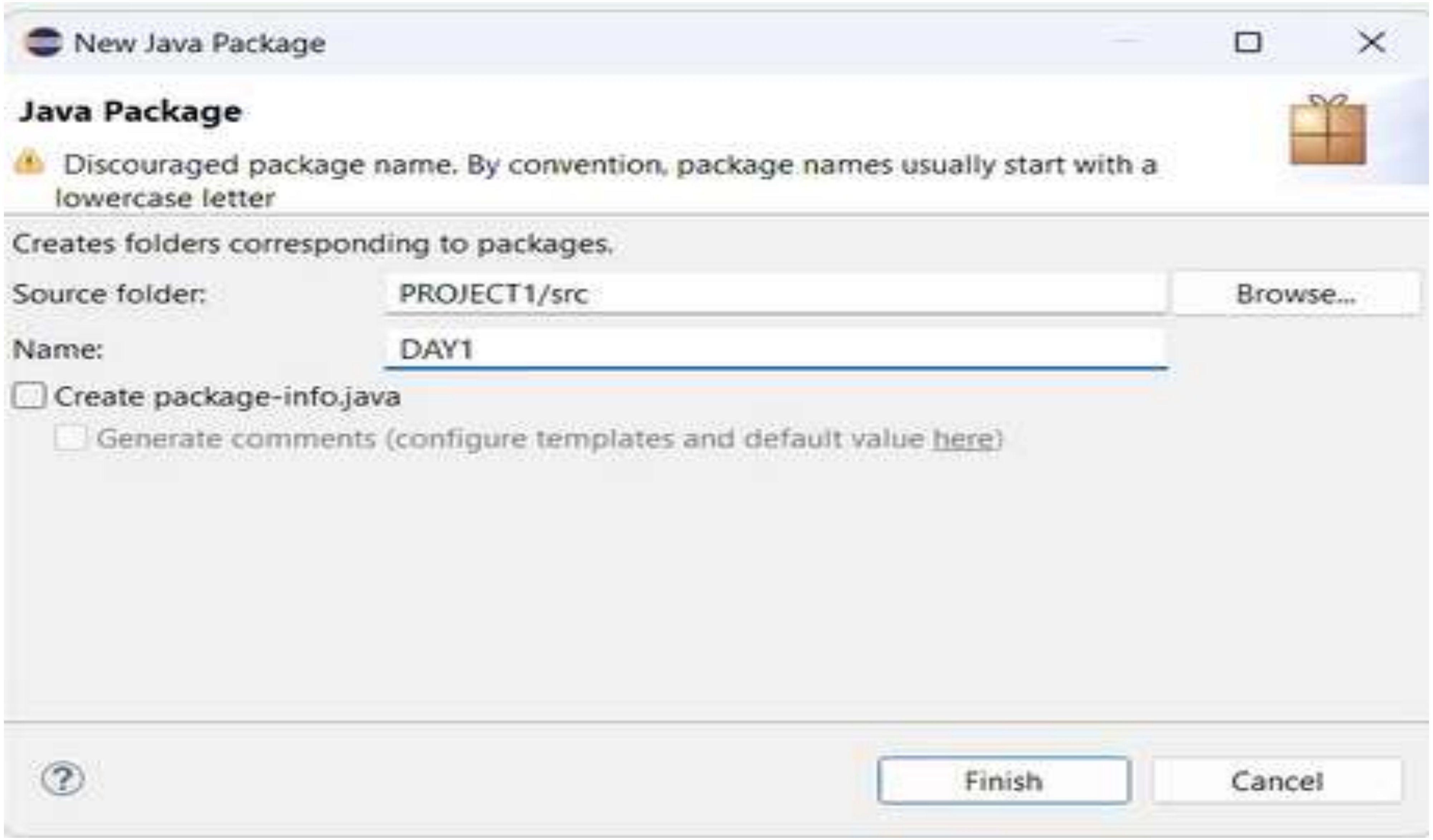
Module name: PROJECT1

Generate comments (configure templates and default value here)

Steps for creating package in project

- Right click on project
- Goto new
- Select package
- Specify package name
- Click next
- Click finish


Package Explorer
PROJECT1
New
Go Into
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Close Project
Assign Working Sets...
Coverage As
Run As
Debug As
Java Project
Project...
Package
Class
Interface
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Example...
Other... Ctrl+N



Java Package

Discouraged package name. By convention, package names usually start with a lowercase letter

Creates folders corresponding to packages.

Source folder: PROJECT1/src

Name: DAY1

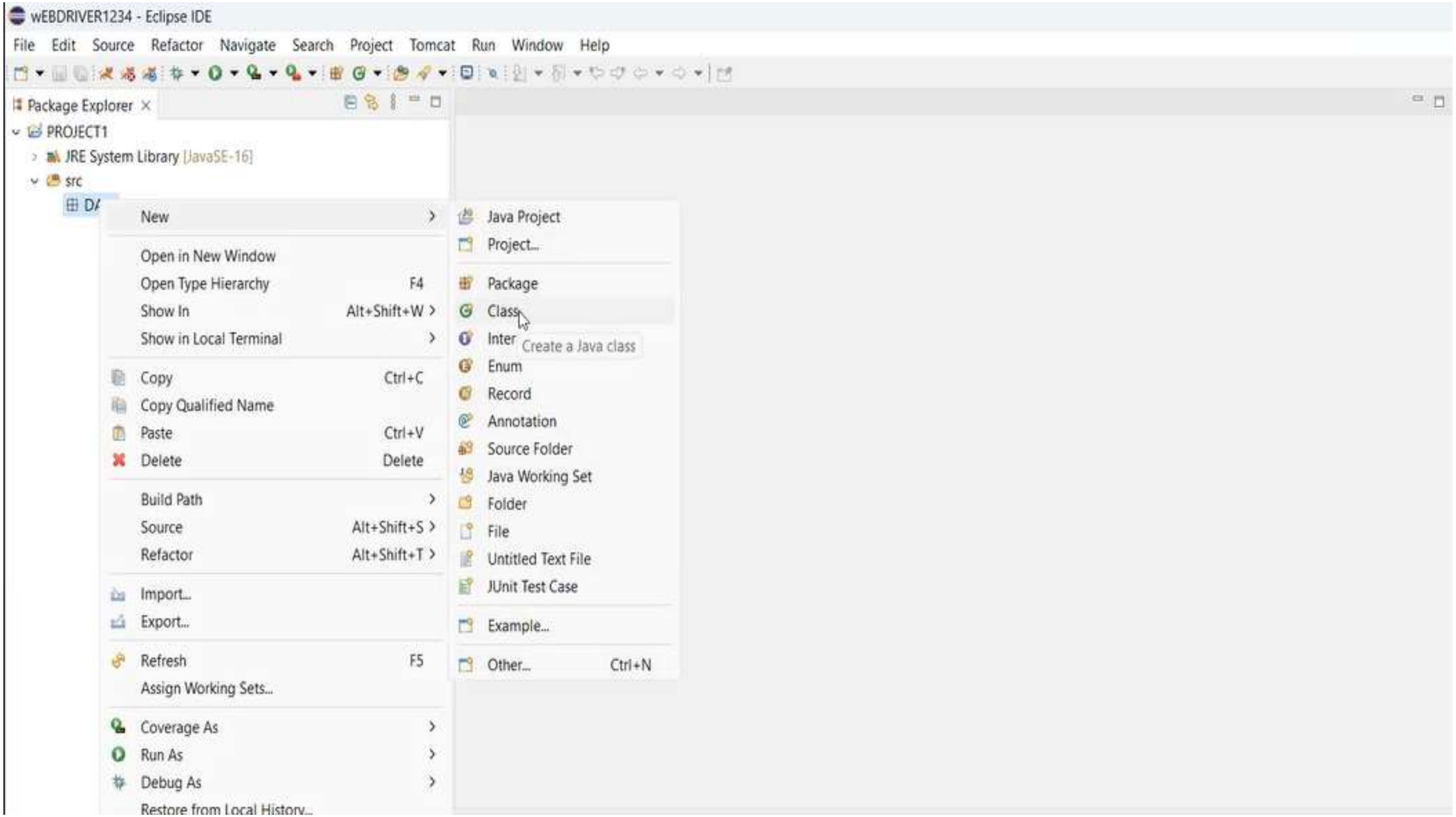
Create package-info.java

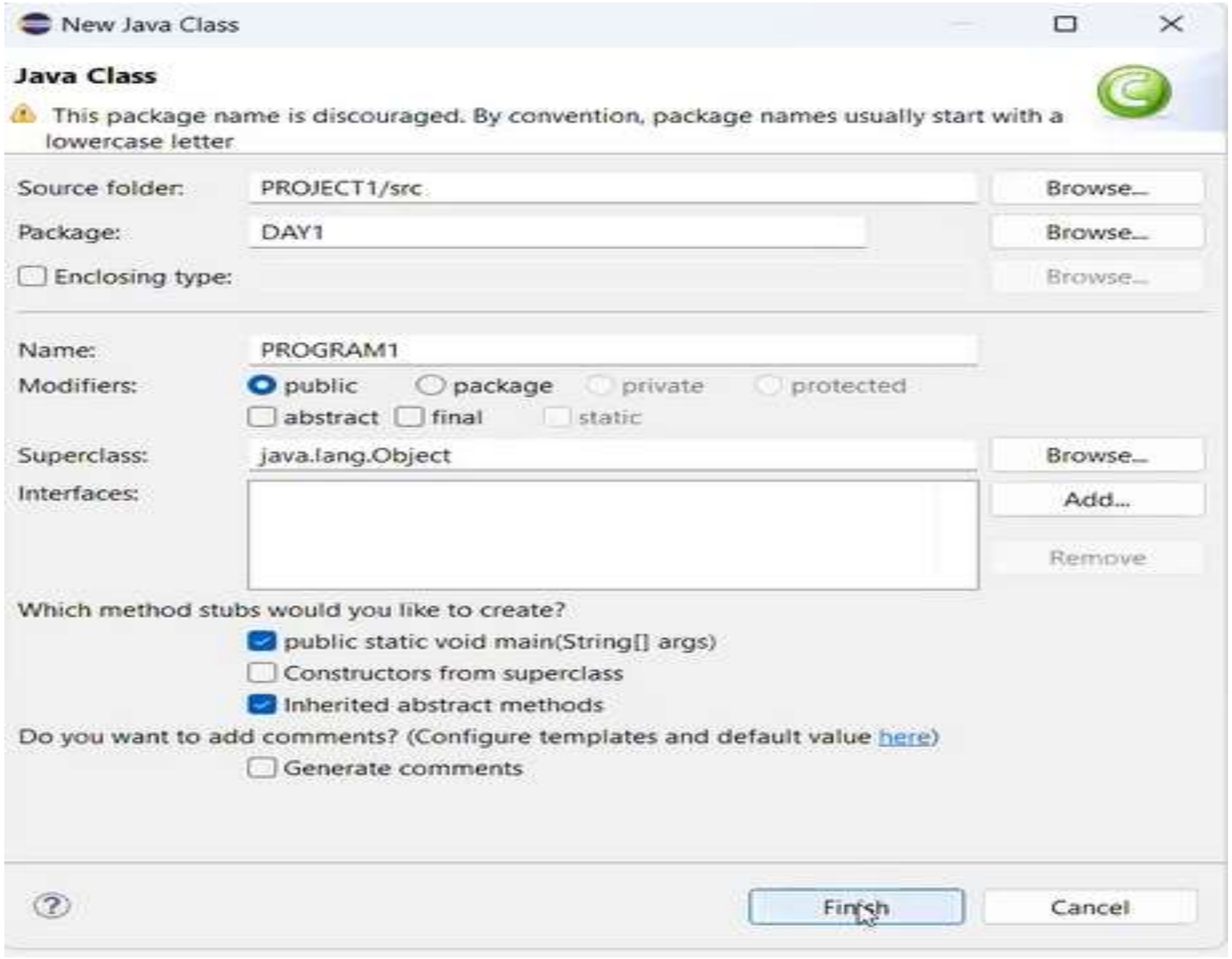
Generate comments (configure templates and default value here)

Steps for creating class in a package

Right click on package

- Goto new
- Select class
- Specify class name
- Select a check box (public static void main(string[] args)
- Click next
- Click finish


wEBDRIVER1234 - Eclipse IDE
File Edit Source Refactor Navigate Search Project Tomcat Run Window Help
Package Explorer
PROJECT1
JRE System Library [JavaSE-16]
src
D/
New
Open in New Window
Open Type Hierarchy F4
Show In Alt+Shift+W
Show in Local Terminal
Copy Ctrl+C
Copy Qualified Name
Paste Ctrl+V
Delete Delete
Build Path
Source Alt+Shift+S
Refactor Alt+Shift+T
Import...
Export...
Refresh F5
Assign Working Sets...
Coverage As
Run As
Debug As
Restore from Local History...
Java Project
Project...
Package
Class
Inter
Create a Java class
Enum
Record
Annotation
Source Folder
Java Working Set
Folder
File
Untitled Text File
JUnit Test Case
Example...
Other... Ctrl+N



New Java Class

Java Class

This package name is discouraged. By convention, package names usually start with a lowercase letter

Source folder: PROJECT1/src

Package: DAY1

☐ Enclosing type:

Name: PROGRAM1

Modifiers: [✓ public] package private protected

☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

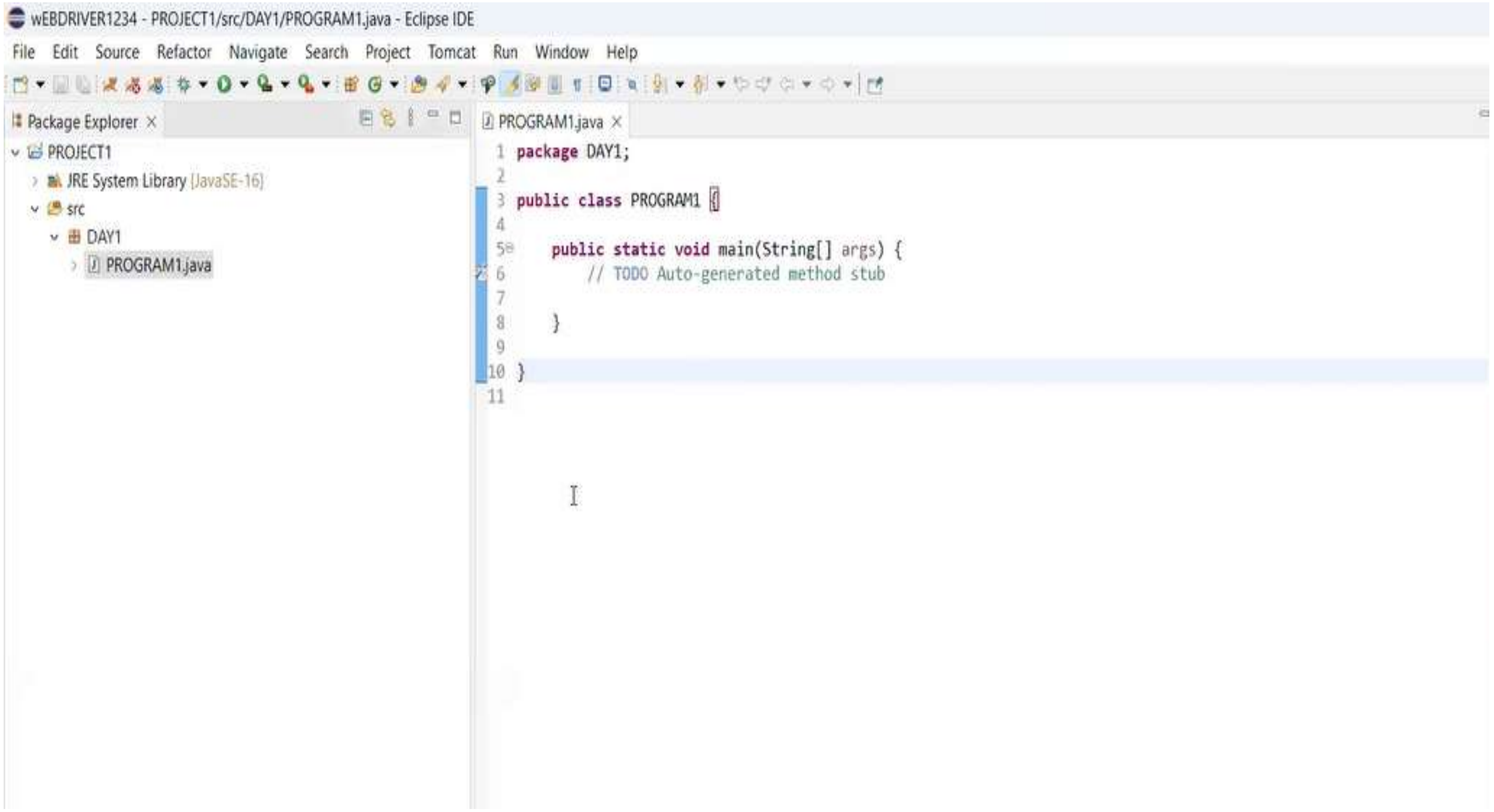
☑ public static void main(String[] args)

☐ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)

☐ Generate comments


wEBDRIVER1234 - PROJECT1/src/DAY1/PROGRAM1.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Tomcat Run Window Help
Package Explorer
PROJECT1
JRE System Library [JavaSE-16]
src
DAY1
PROGRAM1.java
PROGRAM1.java

```java
package DAY1;

public class PROGRAM1 {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

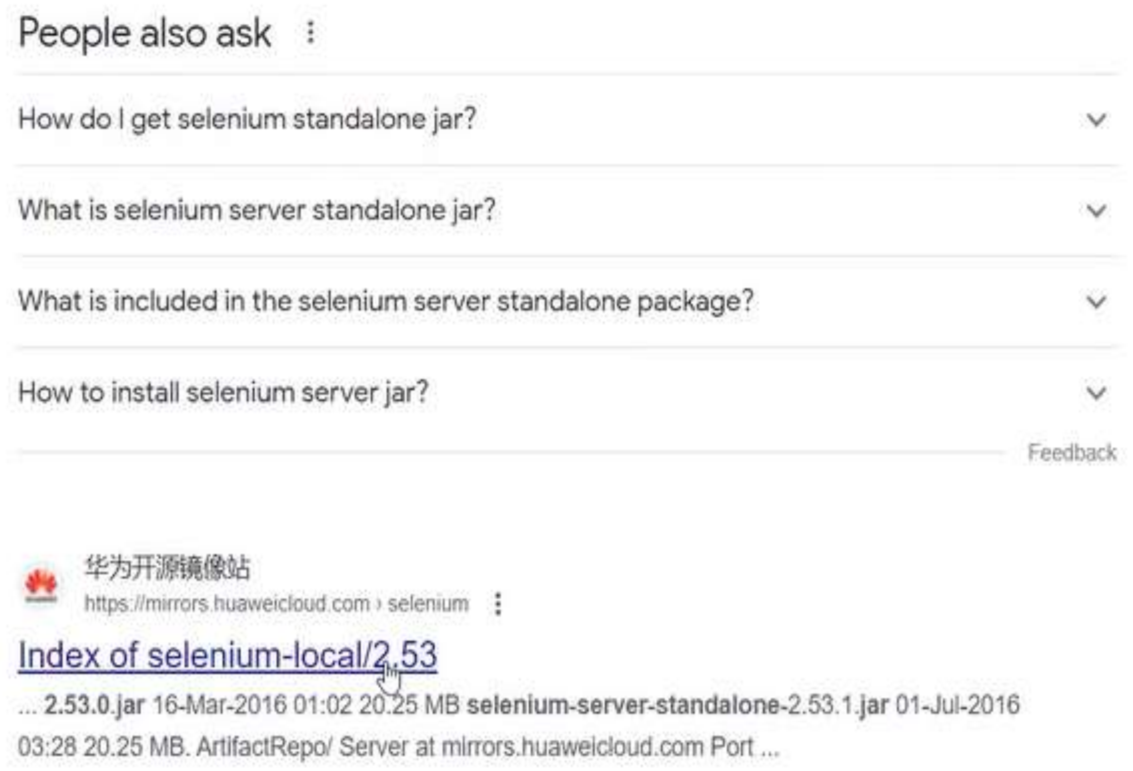
Import jar files into project

- Right click on project
- Goto properties
- Java build path
- Click on libraries
- Select class path
- Click on add external jar files
- Select the required jar file (selenium-server-standalone-2.53.0) jar file
- Click next
- Click finish

- For downloading jar files:
- Go to google search and type

selenium-server-standalone-2.53.0.jar

scroll down the page, there is a link “ Index of selenium-local/2.53” click on that link



People also ask

How do I get selenium standalone jar?

What is selenium server standalone jar?

What is included in the selenium server standalone package?

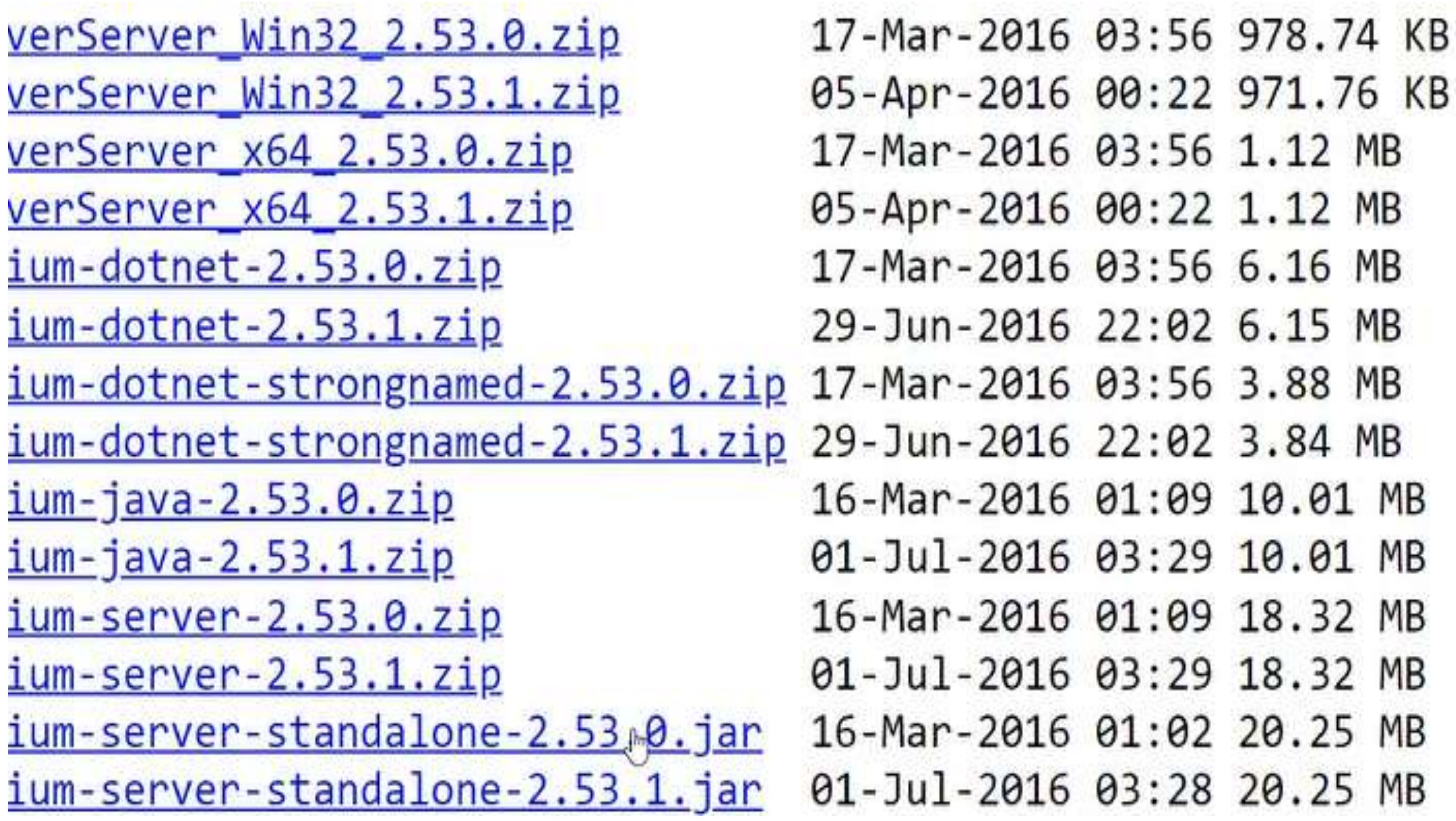
How to install selenium server jar?

Feedback

华为开源镜像站
https://mirrors.huaweicloud.com › selenium

Index of selenium-local/2.53

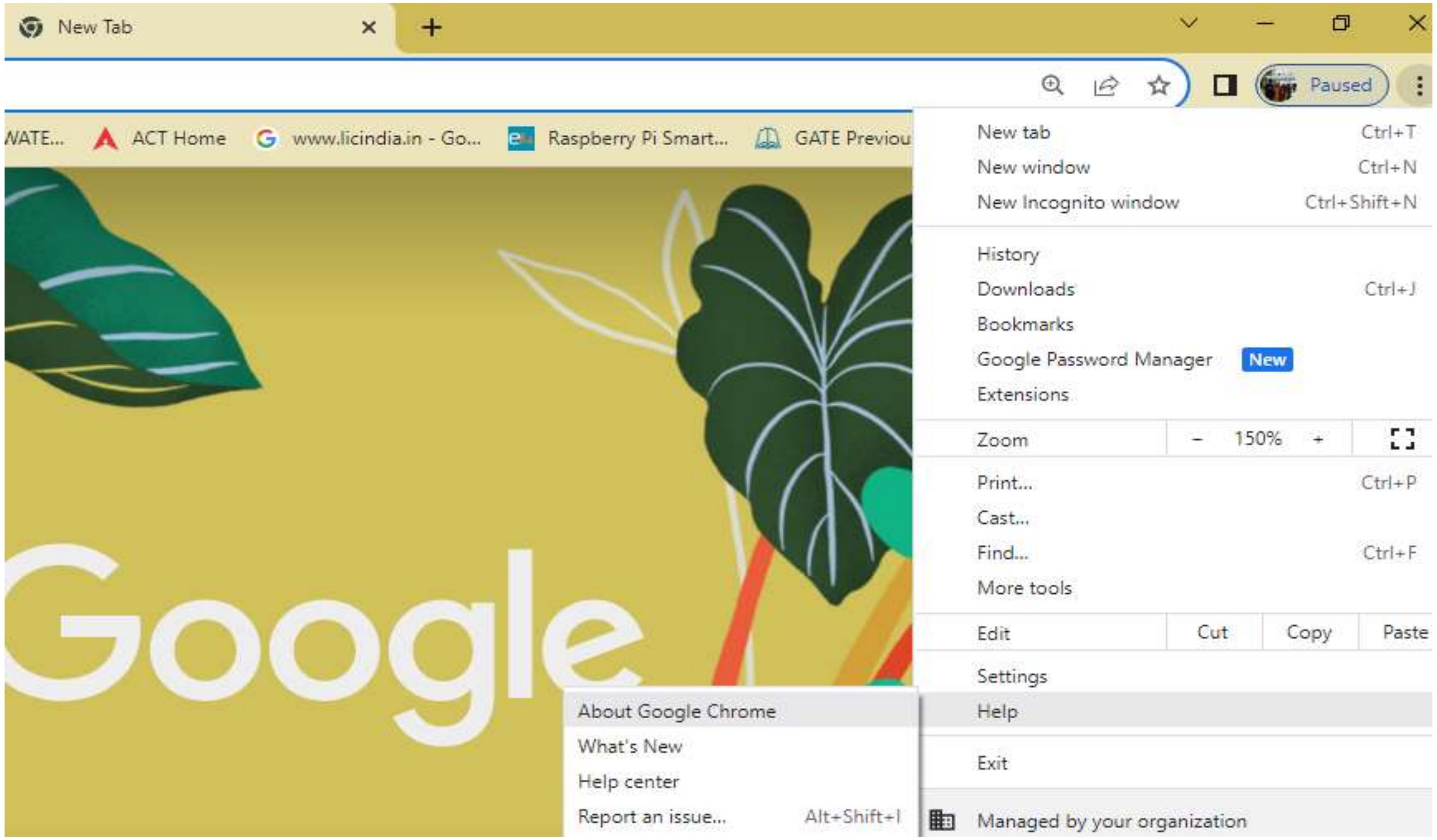
... 2.53.0.jar 16-Mar-2016 01:02 20.25 MB selenium-server-standalone-2.53.1.jar 01-Jul-2016 03:28 20.25 MB. ArtifactRepo/ Server at mirrors.huaweicloud.com Port ...

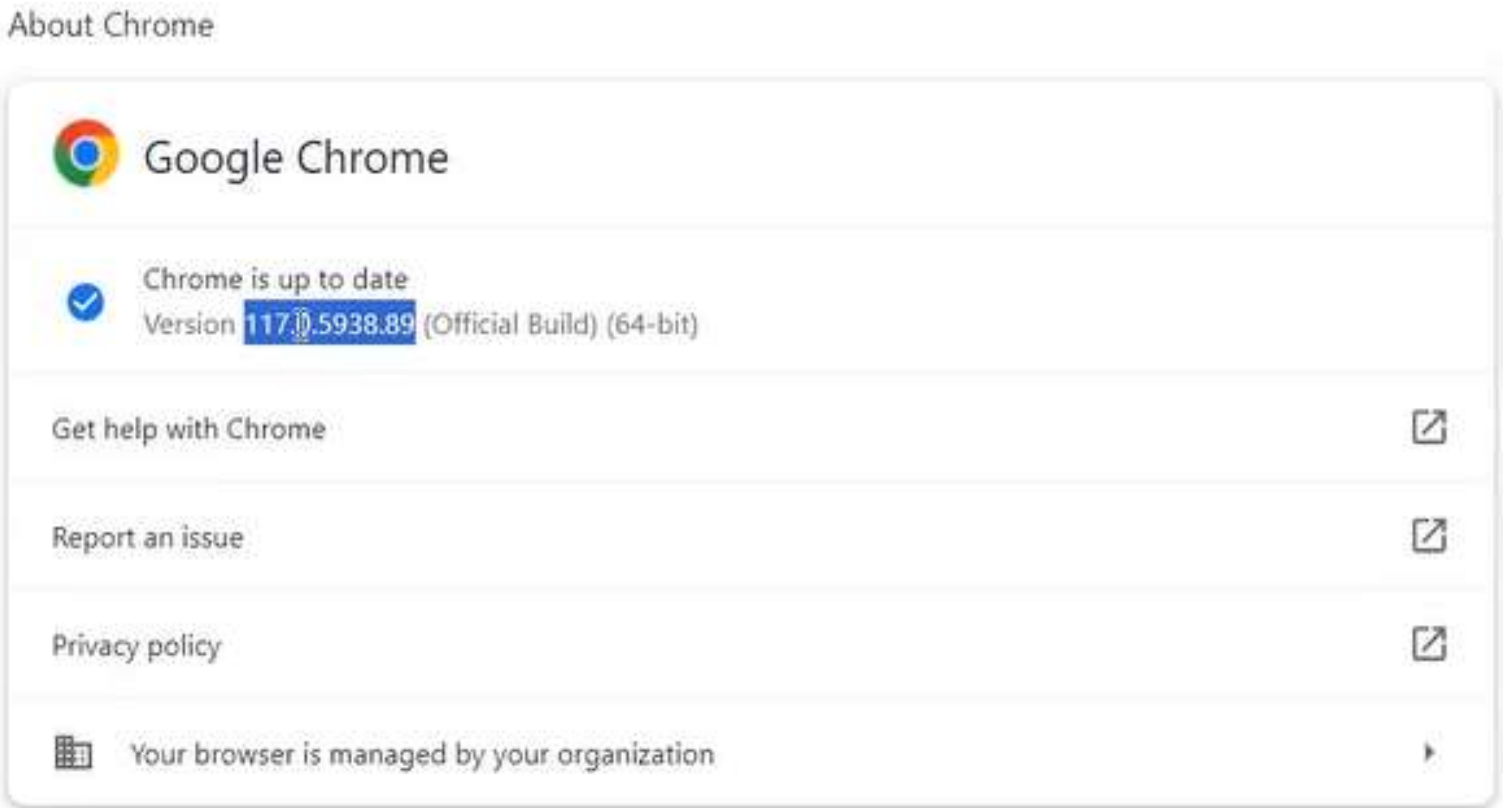


```
verServer_Win32_2.53.0.zip               17-Mar-2016 03:56 978.74 KB
verServer_Win32_2.53.1.zip               05-Apr-2016 00:22 971.76 KB
verServer_x64_2.53.0.zip                 17-Mar-2016 03:56 1.12 MB
verServer_x64_2.53.1.zip                 05-Apr-2016 00:22 1.12 MB
ium-dotnet-2.53.0.zip                    17-Mar-2016 03:56 6.16 MB
ium-dotnet-2.53.1.zip                    29-Jun-2016 22:02 6.15 MB
ium-dotnet-strongnamed-2.53.0.zip        17-Mar-2016 03:56 3.88 MB
ium-dotnet-strongnamed-2.53.1.zip        29-Jun-2016 22:02 3.84 MB
ium-java-2.53.0.zip                      16-Mar-2016 01:09 10.01 MB
ium-java-2.53.1.zip                      01-Jul-2016 03:29 10.01 MB
ium-server-2.53.0.zip                    16-Mar-2016 01:09 18.32 MB
ium-server-2.53.1.zip                    01-Jul-2016 03:29 18.32 MB
ium-server-standalone-2.53.0.jar         16-Mar-2016 01:02 20.25 MB
ium-server-standalone-2.53.1.jar         01-Jul-2016 03:28 20.25 MB
```

Downloading chrome webdriver

- Based on the browser version we need to download the drivers to work on web driver in eclipse





About Chrome

Google Chrome

Chrome is up to date
Version 117.0.5938.89 (Official Build) (64-bit)

Get help with Chrome

Report an issue

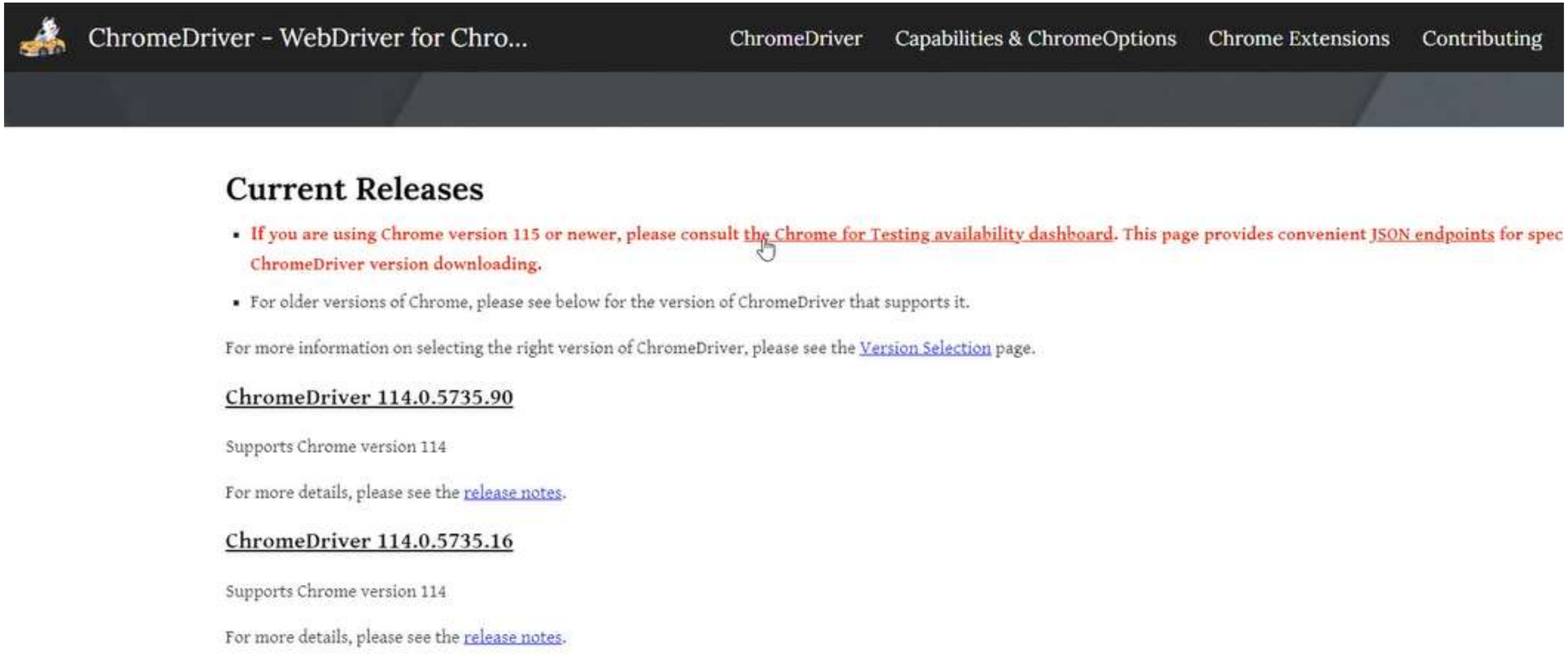
Privacy policy

Your browser is managed by your organization

- Goto chrome browser.
- Type “chromedriver download”
- Click on the link

    ChromeDriver – WebDriver for Chrome

It opens chrome webdrivers page as shown:-


ChromeDriver - WebDriver for Chro...

ChromeDriver Capabilities & ChromeOptions Chrome Extensions Contributing

Current Releases

- **If you are using Chrome version 115 or newer, please consult the Chrome for Testing availability dashboard. This page provides convenient JSON endpoints for spec ChromeDriver version downloading.**
- For older versions of Chrome, please see below for the version of ChromeDriver that supports it.

For more information on selecting the right version of ChromeDriver, please see the Version Selection page.

ChromeDriver 114.0.5735.90

Supports Chrome version 114

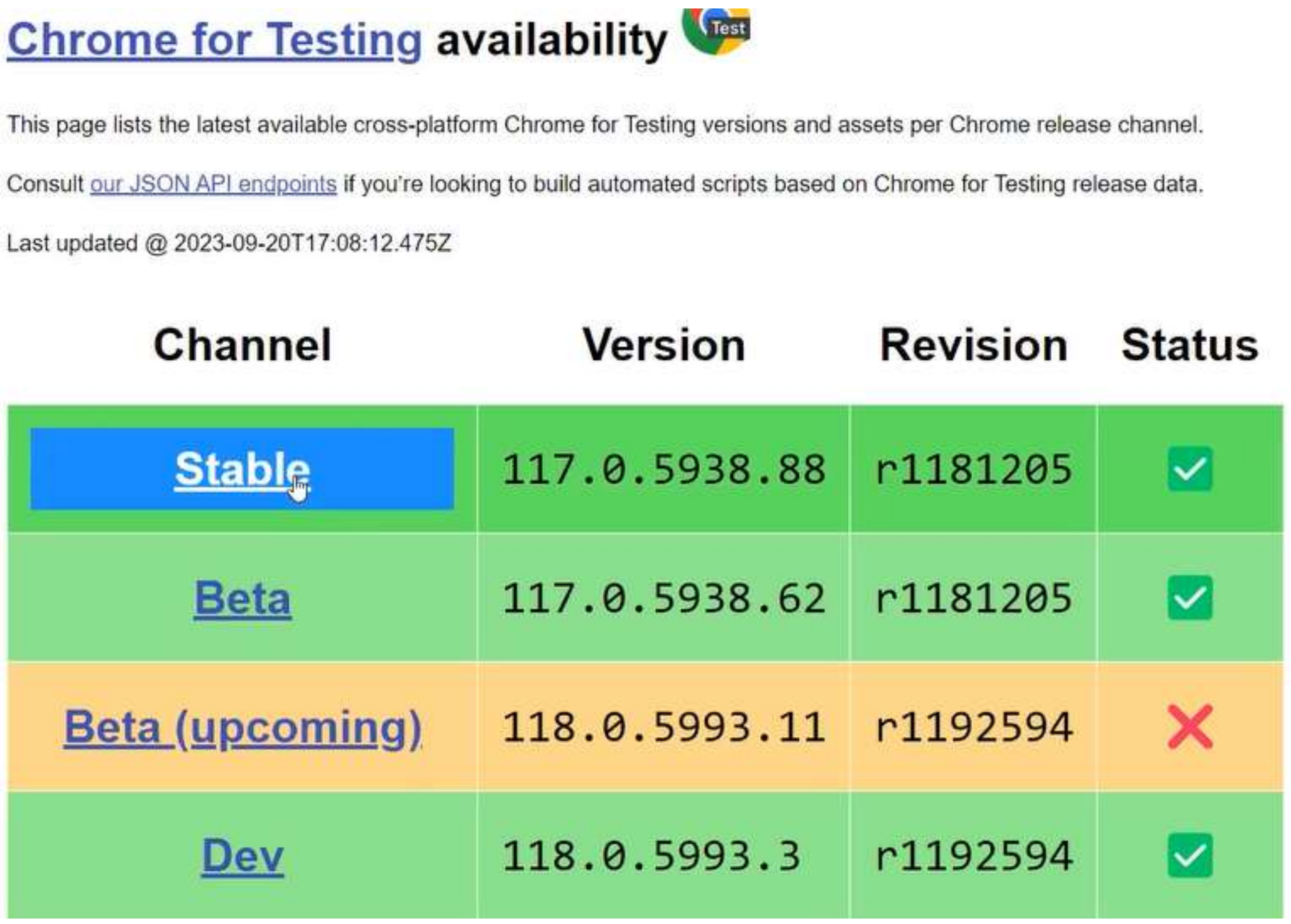
For more details, please see the release notes.

ChromeDriver 114.0.5735.16

Supports Chrome version 114

For more details, please see the release notes.

Click on the stable version



Chrome for Testing availability

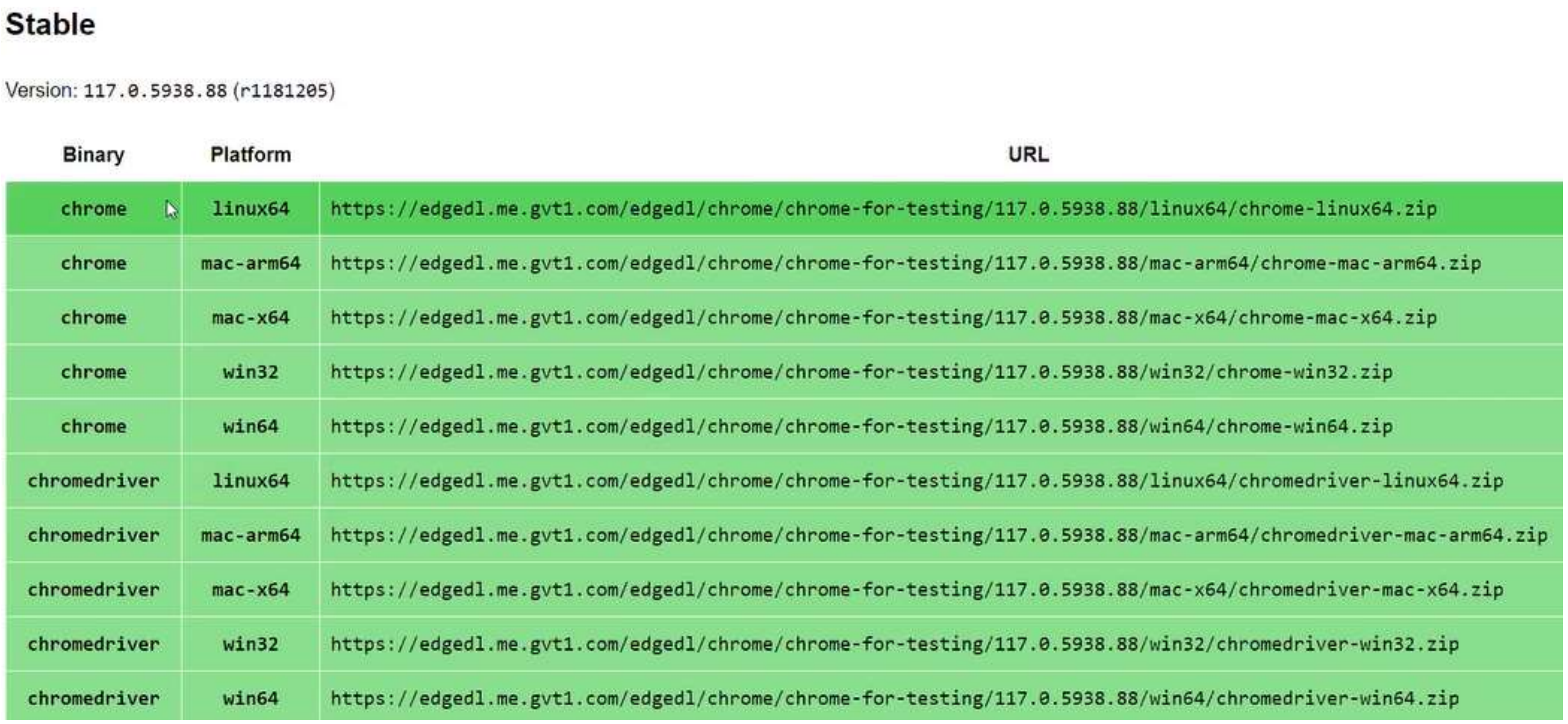
This page lists the latest available cross-platform Chrome for Testing versions and assets per Chrome release channel.

Consult our JSON API endpoints if you're looking to build automated scripts based on Chrome for Testing release data.

Last updated @ 2023-09-20T17:08:12.475Z

Channel	Version	Revision	Status
Stable	117.0.5938.88	r1181205	✓
Beta	117.0.5938.62	r1181205	✓
Beta (upcoming)	118.0.5993.11	r1192594	✗
Dev	118.0.5993.3	r1192594	✓

The following page opens:

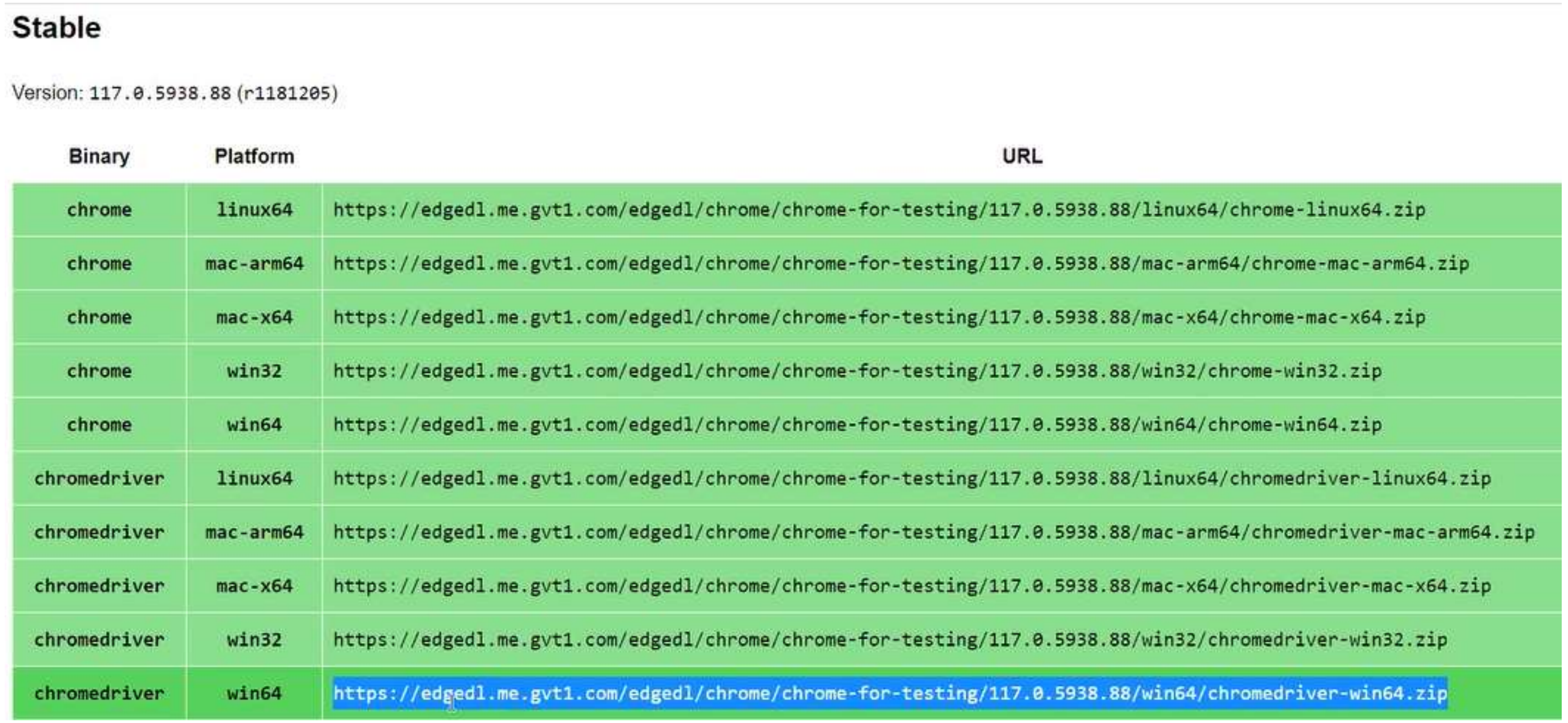


Stable

Version: 117.0.5938.88 (r1181205)

Binary	Platform	URL
chrome	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/linux64/chrome-linux64.zip
chrome	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-arm64/chrome-mac-arm64.zip
chrome	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-x64/chrome-mac-x64.zip
chrome	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win32/chrome-win32.zip
chrome	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win64/chrome-win64.zip
chromedriver	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/linux64/chromedriver-linux64.zip
chromedriver	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-x64/chromedriver-mac-x64.zip
chromedriver	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win32/chromedriver-win32.zip
chromedriver	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win64/chromedriver-win64.zip

Select the url specified and copy , paste it in browser and press ‘ Enter key’. The chrome driver will be downloaded

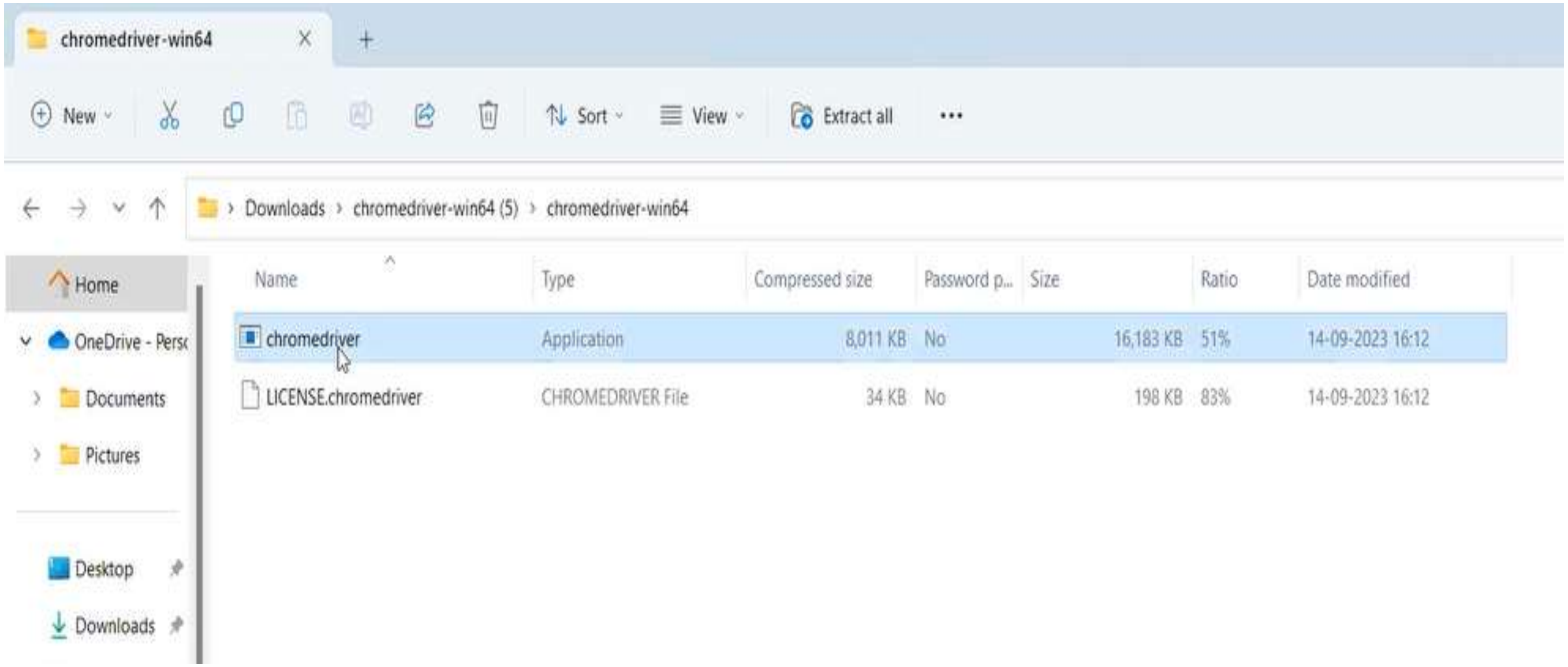


Stable

Version: 117.0.5938.88 (r1181205)

Binary	Platform	URL
chrome	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/linux64/chrome-linux64.zip
chrome	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-arm64/chrome-mac-arm64.zip
chrome	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-x64/chrome-mac-x64.zip
chrome	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win32/chrome-win32.zip
chrome	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win64/chrome-win64.zip
chromedriver	linux64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/linux64/chromedriver-linux64.zip
chromedriver	mac-arm64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-arm64/chromedriver-mac-arm64.zip
chromedriver	mac-x64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/mac-x64/chromedriver-mac-x64.zip
chromedriver	win32	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win32/chromedriver-win32.zip
chromedriver	win64	https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/117.0.5938.88/win64/chromedriver-win64.zip

- Extract the downloaded driver and copy the chrome driver icon into your folder



Name	Type	Compressed size	Password p...	Size	Ratio	Date modified
chromedriver	Application	8,011 KB	No	16,183 KB	51%	14-09-2023 16:12
LICENSE.chromedriver	CHROMEDRIVER File	34 KB	No	198 KB	83%	14-09-2023 16:12

- Example:

  Opening chrome browser

  Navigate to google page

- Maximize the window
- Get the title and url of the page
- Close the browser

```
• package DAY1;
• import org.openqa.selenium.chrome.ChromeDriver;
• public class PROGRAM1 {
•         public static void main(String[] args) {
•                 // TODO Auto-generated method stub chrome driver
•         System.setProperty("webdriver.chrome.driver","C://chromedriver.exe");
•                 ChromeDriver a=new ChromeDriver();
•                 a.get("http://www.google.com");
•                 a.manage().window().maximize();
```

- System.out.println(a.getTitle());
- System.out.println(a.getCurrentUrl());
- a.close();
- }
- }

-
-
-
- }
-
- }

- METHODS AND LOCATORS:

Get():invoke an applciaiton or to call an application

Syntax:

- Obj.get(:http://www.google.com”);

Navigate().to(): to invoke or call an application

- Obj.navigate().to(http://www.google.com);
- Maximizing (): to max the current window
- Syntx:
- Obj.manage().window().maximize();

Gettitle(): to get the current page title

- Syntax:
- System.out.println(obj.gettitle());
- String a=d.gettitle();
- S.o.p(a);

Getcurrenturl(): to get the current page url

- Syntax:
- S.o.p(obj.getcurrenturl());
- String b=obj.getcurrenturl());
- System.out.println(b)';

Forward: to move to next window

- Syntax:
- Obj.navigate().forward();

Back(): to move to previous page

- Syntax:
- Obj.navigate().back();

Refresh(): to refresh the current page

- Syntax
- Obj.navigate().refresh();

Close():to close the current page

- Syntax:
- Obj.close();

Quit(): to close all the browsers

- Syntax:
- Obj.quit();

findElement() : to capture a particular element from the current window

- syntax:
- obj.findElement(By.locatorname(objename)).operations();

findElements() to capture multiple elements from the current window

- ex:
- dropdown , links, radio buttons, checkboxes
- syntax:
- obj.findElements(By.locatorname(“objname”)).opertions();

getpagesource(): to capture a particular text present in the window

- syntax:
- S.o.p(obj.getpagesource());

Basic Exercises on Automation Script:

1.Write a Script to open and close the browser based on user input.

```
 public class Demo {
 public static void main(String[] args) throws InterruptedException {
 Scanner sc = new Scanner(System.in);
System.out.println("Enter browser Name:");
String browser = sc.nextLine();
WebDriver driver = null;
if(browser.equals("Firefox")) {
System.setProperty("webdriver.gecko.driver","./drivers/geckodriver.exe");
driver = new FirefoxDriver(); }
else if(browser.equals("Chrome")) {
System.setProperty("webdriver.chrome.driver","./drivers/chromedriver.exe");
driver = new ChromeDriver(); }
else { System.out.println("Invalid browser"); }
Thread.sleep(2000);
driver.close(); } }
```

- WEB DRIVER METHODS: Methods of WebDriver Interface:

1. get()	To enter the url
2.getTitle()	To get the title of current web page
3.getCurrentUrl()	To get the url of current web page
4.getPageSource()	To get the page source of current web page
5.findElement()	To get single webElements
6.findElements()	To get multiple webElements
7.getWindowHandle()	To get the id of parent window
8.getWindowHandles()	To get the id of All windows
9.switchTo()	Used to switch one window to other window
10. manage()	1. Window 2. Cookies
11.navigate()	1. Enter the URL 2. Navigate to previous page 3. Navigate to next page 4. Refresh current web page
12.close()	To close the current/parent browser
13.quit()	To close all the browsers opened by selenium

Write a script for the following:

- Open the browser
- Delete all cookies
- Set size of the window
- Set position of the window
- Maximize the window

```
• public class Demo {
public static void main(String[] args) throws InterruptedException {
System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");
WebDriver driver = new ChromeDriver();                    //To open the browser
Thread.sleep(2000);
driver.manage().deleteAllCookies();                       //To delete cookies
 Dimension d = new Dimension(500, 500);                   //To set the size of the window
driver.manage().window().setSize(d);
Thread.sleep(2000);
Point p = new Point(250, 250);                            //To set the position of the window
driver.manage().window().setPosition(p);
Thread.sleep(2000);
driver.manage().window().maximize();                       //To maximize the window
} }
```

- NAVIGATION COMMANDS

1. Navigate().to()
2. Refresh()
3. Back()
4. Forword()

Ex.

```
public class Demo {
public static void main(String[] args) throws InterruptedException { //open the browser
System.setProperty("webdriver.chrome.driver", "./drivers/chromedriver.exe");
WebDriver driver = new ChromeDriver(); //To maximize the window
driver.manage().window().maximize();
driver.manage().deleteAllCookies(); //To delete the cookies
driver.get("https://www.google.com/");           //To enter the url
driver.navigate().to("https://www.facebook.com/");
Thread.sleep(1000);
driver.navigate().back(); //To navigate to previous page
Thread.sleep(1000);
driver.navigate().forward(); //To navigate to next page
Thread.sleep(1000);
driver.navigate().refresh(); //Refresh current web page
} }
```

Difference Between get() and navigate():

- get :

It will just enter the URL. After entering the URL it will not allow any statements to execute until the page loads completely

- navigate :

1. It will enter the URL
2. It will navigate to previous page
3. It will navigate to next page
4. It will refresh the current web page

After entering the URL it will not wait until the page loads completely.

WEBELEMENTS METHODS:

- WebElement: Anything which is present on the webpage is called as webelement.

  Ex: text box, link, image, listbox, checkbox etc.

  These Web elements are developed by using HTML.

- HTML stands for HyperText Markup Language.

  The components of HTML are:

i) Tags

ii) Attributes

iii) Text

Before performing action on any elements, we have to perform following steps.

1. Inspect the element
2. Identify/locate the element
3. Find the element
4. Perform the action

Inspect the element:- Fetching the source code of an element is called as inspect the element.

• To inspect the elements, Right click on element click on Inspect element, which will give source code of that element

• In some applications, for security purpose right click option will be disabled.

In such cases

➢ Press F12 which will open developers tool

➢ Select inspect button(mouse icon available on top left corner)

➢ click on the element

- **Methods of WebElement Interface:**

1 sendKeys()	1.To enter the value in textbox 2. To Handle some keyboard action
2 clear()	To clear the textbox value
3 Click()	To click the particular webElement(Button)
4 getCssValue()	To get the color/size/font of the particular webelement
5 findElement()	To get single webElements
6 findElements()	To get multiple webElements
7 getText()	To get the text of the particular webelement
8 getAttribute()	To get the text of the particular attribute(id, name, value…etc.)

9.getTagName()	To get the tagname of particular webelement.
10.getLocation()	To get the X axis and Y axis location of particular webelement.
11.getSize()	To get the size of particular webelement(textbox, text…etc).
12.isDisplayed()	To check whether the particular webelement. is displaying or not(logo, textbox, text…etc)
13.isEnabled()	To check whether the textbox is enabled to enter the text or not.
14.isSelected()	To check whether the radiobutton/dropdown is selected or not.
15.submit()	To click on an element only if the type of the element is submit. Ex: <input type="submit" id="s" value="Submit">

WEB LOCATORS: Static methods which are used to identify the elements which are present the webpage.

All these locators are present in a class called **By** which is an Abstract class.

There are 8 types of locators and all the locators takes argument of type string.

They are:-

1. Id(String)
2. name(String)
3. className(String)
4. tagName(String)
5. linkText(String)
6. partialLinkText(String)
7. cssSelector(String)
8. xpath(String)

Adactin hotel login using id/name:

```
public class Login {
public static void main(String[] args) throws Throwable {
System.setProperty("webdriver.chrome.driver", "C:\\Users\\chromedriver.exe");
WebDriver driver = new ChromeDriver();
driver.get("https://adactin.com/HotelApp/index.php");
driver.manage().window().maximize();
WebElement x = driver.findElement(By.id("username"));
                 x.sendKeys("venkat");
WebElement x1 = driver.findElement(By.name("password"));
                x1.sendKeys("venkat@123445");
WebElement x2 = driver.findElement(By.id("login"));
                x2.click();
}
}
```

getAttribute() and getText(): It is a method, used to print the value whatever you gave in the text box

Example program:

```
public class Ex5 {
public static void main(String[] args) {
System.setProperty("webdriver.gecko.driver","C:/Users/Selenium/geckodriver.
exe");
WebDriver driver=new FirefoxDriver();
driver.get("http://www.adactin.com/HotelApp/index.php");
driver.findElement(By.id("username")).sendKeys("venkat16");
driver.findElement(By.id("password")).sendKeys("Karthik");
String s = driver.findElement(By.id("username")).getAttribute("value");
String s1 = driver.findElement(By.id("password")).getAttribute("value");
System.out.println(s);
System.out.println(s1);
}
}
```

Locating Strategies- (By Name)

How to locate a particular web element using the value of its "name" attribute. Now we will try to locate the desired web element by using the value of its "name" attribute. In Selenium, locating a particular web element involves inspection of its HTML codes.

Let us consider a test case in which we will automate the following scenarios:

- Invoke Firefox browser
- Open URL: https://www.testandquiz.com/selenium/testing.html
- Click on the Text Box
- Type the value "Selenium Tutorial"

Follow the steps given below to locate the Text box on the sample webpage.

Open URL: https://www.testandquiz.com/selenium/testing.html

Right click on the Text box and select Inspect Element.

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

Pick the value of name attribute i.e. "firstName".

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

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

- Therefore, for locating the text box on the sample web page we will use the value of its name attribute as:

```
driver.findElement(By.name (<"firstName">))
```

```
1.import org.openqa.selenium.By;
2.import org.openqa.selenium.WebDriver;
3.import org.openqa.selenium.firefox.FirefoxDriver;
//import org.openqa.selenium.remote.DesiredCapabilities;
  public class Name_Test {
   public static void main(String[] args) {
    // System Property for Gecko Driver
        System.setProperty("webdriver.gecko.driver","D:\\GeckoDriver\\geckodriver.exe" );
                // Initialize Gecko Driver using Desired Capabilities Class
        //DesiredCapabilities capabilities = DesiredCapabilities.firefox();
           // capabilities.setCapability("marionette",true);
            WebDriver driver= new FirefoxDriver(capabilities);
             // Click on the textbox and send value
     driver.findElement(By.name("firstName")).sendKeys("Selenium Tutorial");
          }  }
```

- *Right click on the Eclipse code and select **Run As > Java Application**.*
- *Upon execution, the above test script will launch the Firefox browser and automate all the test scenarios.*

- Locating Strategies- (By Class Name)

how to locate a particular web element using the value of its Class attribute.

Let us consider a test case in which we will automate the following scenarios:

- Invoke Chrome browser
- Open URL: https://www.testandquiz.com/selenium/testing.html
- Click on the checkbox value "Automation Testing“.
- Now, we will try to locate the desired web element by using the value of its Class attribute. In Selenium, locating a particular web element involves inspection of its HTML codes.
- Follow the steps given below to locate the Checkbox on the sample web page.
- Open URL: https://www.testandquiz.com/selenium/testing.html
- Right click on the Automation Testing Check box and select Inspect

- It will launch a window containing all the specific codes involved in the development of the Checkbox.
- Pick the value of Class attribute i.e. "Automation".

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

driver.findElement(By.className (<element **class**>))

- Therefore, for locating the Checkbox on the sample web page we will use the value of its Class attribute as:

driver.findElement(By.className (<"Automation">))

- To automate our third test scenario, we need to write the code which will click on the Checkbox value "Automation".

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class Class_Test {
   public static void main(String[] args) {
       // System Property for Chrome Driver
      System.setProperty("webdriver.chrome.driver","D:\\ChromeDriver\\chromedriver.exe
");
       // Instantiate a ChromeDriver class.
     WebDriver driver=new ChromeDriver();
       // Launch Website
     driver.navigate().to("https://www.testandquiz.com/selenium/testing.html");
   //Locate the checkbox by Class Name and check it using click() function
     driver.findElement(By.className("Automation")).click();
  }
 }
```

- Locating Strategies- (By Tag Name)

how to locate a particular web element using its Tag Name.

Let us consider a test case in which we will automate the following scenarios:

- Invoke Firefox browser
- Open URL: https://www.testandquiz.com/selenium/testing.html
- Click on the Text Box
- Type the value "C++ Tutorial"
- Click on the Submit button

- Now we will try to locate the desired web element by using its Tag Name. In Selenium, locating a particular web element involves inspection of its HTML codes.
- Follow the steps given below to locate the Text box on the sample web page.
  - Open URL: https://www.testandquiz.com/selenium/testing.html

  Right click on the Text box and select Inspect Element.

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

  Pick the name of first tag i.e. "input".

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

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

Therefore, for locating the textbox on the sample web page we will use the name of its first Tag Element:

driver.findElement(By.tagName (<"input">))

- Similarly, for locating the Submit button on the sample web page we will use the name of its first Tag element:

```
driver.findElement(By.tagName (<"button">))
```

```java
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
//import org.openqa.selenium.remote.DesiredCapabilities;
public class Tag_Test {
    public static void main(String[] args) {
     // System Property for Gecko Driver
         System.setProperty("webdriver.gecko.driver","D:\\GeckoDriver\\geckodriver.exe" );
     // Initialize Gecko Driver using Desired Capabilities Class
     //DesiredCapabilities capabilities = DesiredCapabilities.firefox();
            //capabilities.setCapability("marionette",true);
            WebDriver driver= new FirefoxDriver(capabilities);
     // Click on the textbox and send value
    driver.findElement(By.tagName("input")).sendKeys("C++ Tutorial");
       // Click on the Submit button using click() command
        driver.findElement(By.tagName("button")).click();
       } }
```

Locating Strategies- (By Link Text)

- how to locate a particular web element through its Link Text.
- Let us consider a test case in which we will automate the following scenarios:

Invoke Firefox browser

Open URL: https://www.testandquiz.com/selenium/testing.html

Click on the link text "This is a Link" on the sample web page.

Now, we will try to locate the desired web element through its Link text. In Selenium, locating a particular web element involves inspection of its HTML codes.

Follow the steps given below to locate the Text box on the sample web page.

- Open URL: https://www.testandquiz.com/selenium/testing.html
- Right click on the "This is a Link" text on the sample web page and select Inspect Element.

- It will launch a window containing all the specific codes involved in the development of the Link Text.
- Pick the value of the Link Text i.e. "This is a link".
- The Java Syntax for locating a web element through its Link Text is written as:

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

- Therefore, for locating the Link Text on the sample web page we will use the value of its Link Text:

```
driver.findElement(By.linkText (<"This is a Link">))
```

- To automate our test scenario, we need to write the code which will click on the Link Text.

  Here is the sample code to click on the Link Text.

```
// Click on the Link Text using click() command
driver.findElement(By.linkText("This is a Link")).click();
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
  //import org.openqa.selenium.remote.DesiredCapabilities;
public class Link_Test {
    public static void main(String[] args) {
          // System Property for Gecko Driver
        System.setProperty("webdriver.gecko.driver","D:\\GeckoDriver\\geckodriver.exe" );
              // Initialize Gecko Driver using Desired Capabilities Class
       //DesiredCapabilities capabilities = DesiredCapabilities.firefox();
            //capabilities.setCapability("marionette",true);
            WebDriver driver= new FirefoxDriver(capabilities);
       // Launch Website
       driver.navigate().to("https://www.testandquiz.com/selenium/testing.html");
  // Click on the Link Text using click() command
    driver.findElement(By.linkText("This is a Link")).click();
     }  }
```

Locating Strategies- (By CSS)

- CSS **stands for Cascading Style Sheets**. It is a Style Sheet Language which is used to describe the look and formatting of a document written in markup language.
- locating web elements through CSS involves use of CSS Selector which identifies an element based on the combination of HTML tag, id, class and attributes.
- In WebDriver, CSS Selector works in **six modes** to identify and locate web elements:

1. Tag and ID
2. Tag and Class
3. Tag and Attribute
4. Tag, Class and Attribute
5. Sub-String Matches

Locating Strategies- (By CSS- Tag and ID)

how to locate a particular web element using CSS - Tag and ID Selector.

- Open URL: https://www.testandquiz.com/selenium/testing.html
- Right click on the Textbox on the sample web page and select Inspect Element.
- Take a note of its Tag and value of its id attribute.
- 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"))
```

- Therefore, for locating the Textbox on the sample web page, we will use the input tag along with the value of its id attribute:

```
driver.findElement(By.cssSelector("input#fname"))
```

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
//import org.openqa.selenium.remote.DesiredCapabilities;
public class SampleOne {
    public static void main(String[] args) {
    System.setProperty("webdriver.gecko.driver","D:\\GeckoDriver\\geckodriver.exe" ); // System Property for
Gecko Driver
         // Initialize Gecko Driver using Desired Capabilities Class
        //DesiredCapabilities capabilities = DesiredCapabilities.firefox();
        //capabilities.setCapability("marionette",true);
        WebDriver driver= new FirefoxDriver(capabilities);
       driver.navigate().to("https://www.testandquiz.com/selenium/testing.html");   // Launch Website
driver.findElement(By.cssSelector("input#fname")).sendKeys("JavaTpoint");   // Click on the textbox and sen
d value
    driver.findElement(By.cssSelector("button#idOfButton")).click();    // Click on the Submit button using click(
) command
     driver.close();     //  Close the Browser
    }
}
```

Locating Strategies- (By XPath)

XPath is a language used for locating nodes in XML documents.

- XPath can be used as a substitute when you don't have a suitable id or name attribute for the element you want to locate.
- XPath allows you to select individual elements, attributes, and some other part of XML documents for specifying location of a particular web element.

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

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

- However, there are different ways of writing dynamic XPaths such as:

1. Using Single Slash

Locating Strategies- (By XPath- Using Single Slash)

how to locate a particular web element by XPath- Using Single Slash.

- Single Slash mechanism is also known as finding elements using Absolute XPath. Single Slash is used to create XPath with absolute XPath i.e. the XPath would be created to start selection from the document node/ Start node/ Parent node.

Using Single Slash/Absolute XPath, we can write the Java code along with the dynamic XPath location as:

```
findElement(By.xpath("html/body/div[1]/div[2]/div[2]/div[1]/form/div[1]/div/div[1]/div/div/input[1]”))
```

Locating Strategies- (By XPath- Using Double Slash)

In this section, you will learn how to locate a particular web element by XPath- Using Double Slash.

- Double Slash mechanism is also known as finding elements using Relative XPath. Double slash is used to create XPath with relative path i.e. the XPath would be created to start selection from anywhere within the document. - Search in a whole page (DOM) for the preceding string.

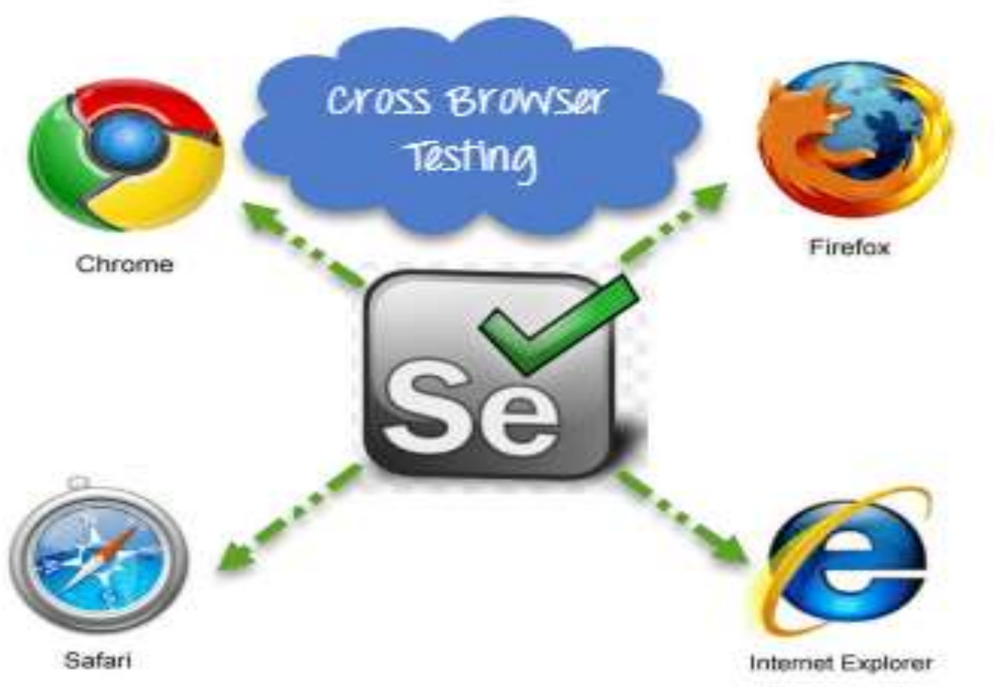
Note: Double Slash is recommended when you don't have a suitable id or name attribute for the element you want to locate.

Using Double Slash/Relative XPath, we can write the Java code along with the dynamic XPath location as:

```
findElement(By.xpath("//form/div[1]/div/div[1]/div/div/input[1]"));
```

Cross browser testing

- *"Cross Browser Testing is a type of testing to verify if an application works across different browsers as expected and degrades gracefully. It is the process of verifying your application's compatibility with different browsers."*



What is Cross Browser Testing?

1) Cross-browser testing is simply what its name means i.e., to test our website or application in multiple browsers- and making sure that it works consistently and as in intended without any dependencies, or compromise in Quality.

2) This is applicable to both web and mobile applications.

3) What kinds of applications undergo this? –

Customer-facing applications are the best choice.

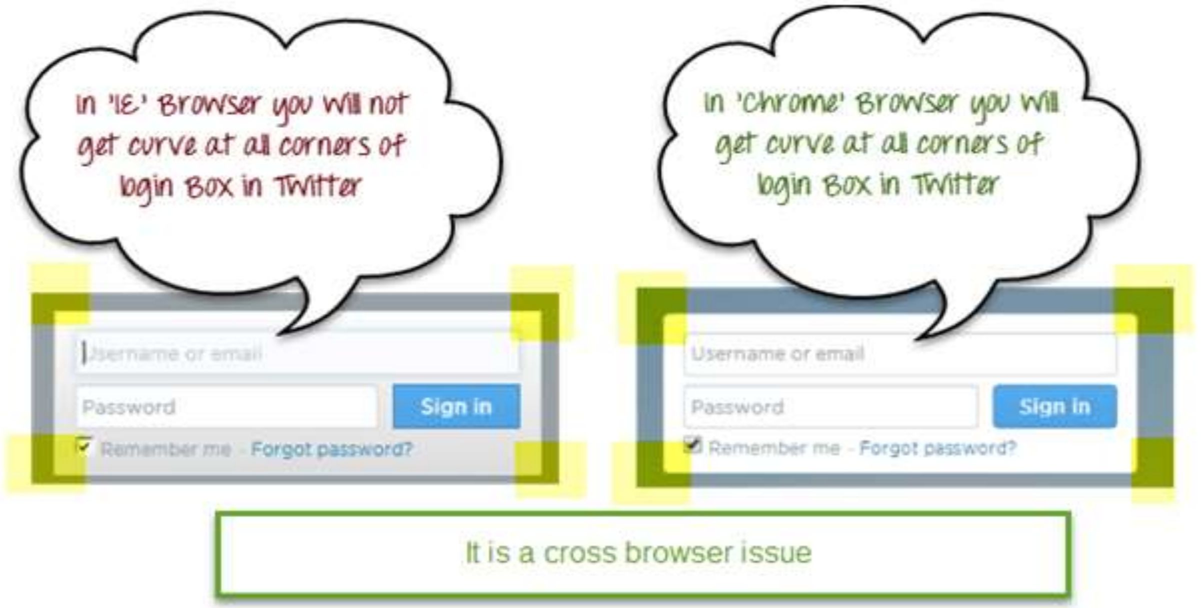
“Aren’t all applications customer-facing?” Well, yes. They are. **However, let us look at an example.**

Application 1: An application developed for a company to internally keep track of its inventory.
Application 2: This is for the end-users to buy products from this company.

- It is apparent that the best idea would be to test Application 2 for browser compatibility testing since it is impossible to control what browsers/platforms/ versions the end-user is going to use.
- On the other hand, if all computers internal to the company use Windows 8 machines with Chrome browser- then there is no need test for anything else with respect to Application 1.

Why do we need Cross Browser Testing?

- Web-based applications are totally different from Windows applications. A web application can be opened in any browser by the end user. For example, some people prefer to open https://twitter.com in **Firefox browser,** while other's can be using **Chrome browser** or **IE**.
- In the diagram below you can observe that in **IE**, the login box of Twitter is not showing curve at all corners, but we are able to see it in Chrome browser.


In 'IE' Browser you will not get curve at all corners of login Box in Twitter
In 'Chrome' Browser you will get curve at all corners of login Box in Twitter
Username or email
Password
Sign in
Remember me · Forgot password?
Username or email
Password
Sign in
Remember me · Forgot password?
It is a cross browser issue

- So we need to ensure that the web application will work as expected in all popular browsers so that more people can access it and use it.
- This can be fulfilled with Cross Browser Testing of the product.

Some of the Cross Browser Issues are:-

1.Font size mismatch in different browsers.

2.JavaScript implementation can be different.

3.CSS,HTML validation difference can be there.

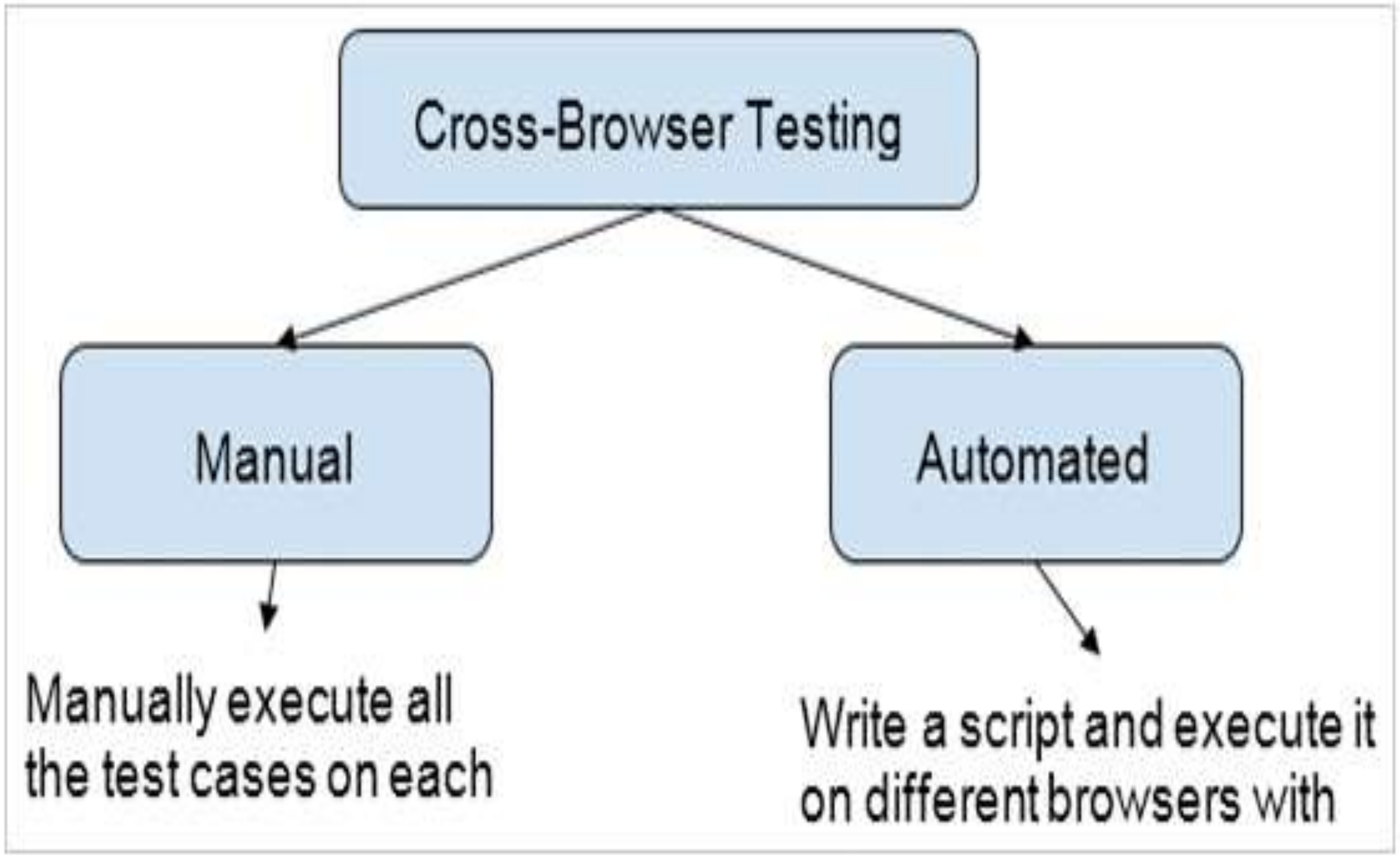
4.Some browser still not supporting HTML5.

5.Page alignment and div size.

6.Image orientation.

7.Browser incompatibility with OS. Etc.

How to Perform Cross Browser Testing?


Cross-Browser Testing
Manual
Automated
Manually execute all the test cases on each
Write a script and execute it on different browsers with

Manual Method

- In this case, a business identifies the browsers that the application must support. Testers then re-run the same test cases using different browsers and observe the application’s behavior and report bugs if any.
- In this type of testing, it is not possible to cover many browsers and also, the application might not be tested on major browser versions.
- Also, performing cross-browser check manually is costly and time-consuming too.

Automated Method

- Cross-browser testing is basically running the same set of test cases multiple times on different browsers.
- This type of repeated task is best suited for automation. Thus, it’s more cost and time effective to perform this testing by using tools.

Lot of Tools are Available in the market to do cross browser testing :

#1) BitBar

#2) TestGrid

#3) Selenium

#4) BrowserStack

#5) Browserling

#6) LambdaTest

When is Cross Browser Testing Done?

Depending on the role and workflow, we could be running cross-browser tests:

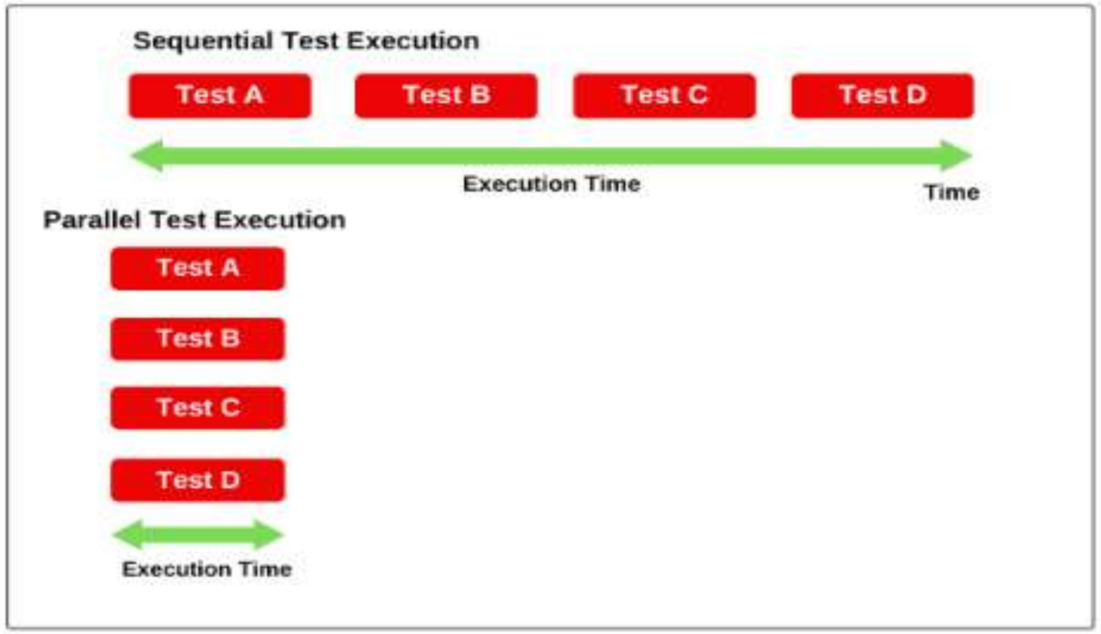
- **During Development:** Developers in Continuous Integration pipelines test new features to make sure they’re cross-browser compatible before pushing the changes to production.
- **In Staging/Pre-Release:** QA teams do this for every Release Candidate to make sure that no browser compatibility issues crop up in the latest version of the website.

Who Does Cross Browser Testing?

- The answer is: Anyone who designs/develops for the Open Web.
- In general, QA teams execute test scenarios on multiple browsers to make sure the build meets browser compatibility benchmarks.
- UI teams run cross browser tests to find out how the website front-end performs on different devices and orientations.

PARALLEL TESTING:

- As teams move towards continuous development and delivery model for the software world, they are exploring solutions to increase their pace of delivery. QA teams try to this by reducing the testing time.
- However, if teams move from sequential testing to parallel testing, teams can achieve maximum coverage with reduced execution time. This new era of the continuous testing model, along with parallel testing, helps drive this approach.



What is Parallel Testing (also referred to as Parallel Test Execution)?

- In parallel testing, we test different modules or applications on multiple browsers in parallel rather than one by one.
- The parallel test execution is different from sequential testing, where we test different modules or functionalities one after the other. Even in the case of testing applications on multiple browsers, tests are performed sequentially on various browsers. This approach of testing is very time-consuming.
- Parallel testing helps to reduce execution time and efforts and results in a faster time to delivery. It proves to be handy specifically in case of cross browser testing, compatibility testing, localization, and internalization testing. In a scenario where we have two versions of software available, and we need to check its stability and compatibility, we can run the two versions simultaneously and find issues at a much faster rate.

Parallel Testing using TestNG and Selenium

- **TestNG** is a testing framework for Java that helps to organize tests in a structured way and enhances maintainability and readability to the scripts. TestNG has made it easier for automation testers owing to its large feature set. One of which is parallel testing or parallel execution. TestNG provides an auto-defined XML file, where one can set the parallel attribute to method/tests/classes and by using the concept of multi-threading of Java, one can set the number of threads, one wants to create for parallel execution.

- **Web/HTML Elements**

Link – click – It redirects
Edit box – Enter a value – It holds the value
Button – click – It submits
Dropdown box – select an item – It holds the item
List Box – Select one or more items – It holds one or more items
Combo box (Dropdown box + edit box) – select an item or type an item
Radio button – click – It selects an option
Checkbox – click – check or uncheck
Text – Information/display
Image (general image, image link, and image button)
TextArea – Enter text –
Web Table/HTML Table
etc.

Working on web elements

Handling Links

- **Types of Links in the web:**
  a. Text Links (Ex: ‘Gmail’ link in Google home page)
  b. Image Links (Ex: ‘TestBirds’ link in selenium.dev home page)
  c. Internal Links
  d. External Links
  e. MailTo Links
  f. Broken Links

- ***Manual Actions:***

- Check the Displayed status
- Check the Enabled status
- Click
- Return Link name

Selenium WebDriver Test Steps:

```
System.setProperty(“webdriver.chrome.driver”, “D://chromedriver.exe”);
WebDriver driver = new ChromeDriver();
driver.manage().window().maximize();

driver.get(“https://www.google.com/”);

WebElement gmail = driver.findElement(By.linkText(“Gmail”));

boolean status = gmail.isDisplayed();
System.out.println(status);                    //true

status = gmail.isEnabled();
System.out.println(status);                    //true

String linkName = gmail.getText();
System.out.println(linkName);

gmail.click();
Thread.sleep(3000);

driver.close();
```

- **Handling Check box**
- ***Manual Actions:***
- • Check Displayed status
  • Check Enabled status
  • Check Selected status
  • Click (Check/Uncheck)

- ***Selenium WebDriver Test Steps:***
- System.setProperty(“webdriver.chrome.driver”, “D://chromedriver.exe”);
  WebDriver driver = new ChromeDriver();
  driver.manage().window().maximize();
- driver.get(“https://gcreddy.com/project/create_account.php”);
- boolean status = driver.findElement(By.name(“newsletter”)).isDisplayed();
  System.out.println(“Displayed status is: “+status); //true
- status = driver.findElement(By.name(“newsletter”)).isEnabled();
  System.out.println(“Enabled status is: “+status); //true
- status = driver.findElement(By.name(“newsletter”)).isSelected();
  System.out.println(“Intial Selected status is: “+status); //false
- driver.findElement(By.name(“newsletter”)).click();
- status = driver.findElement(By.name(“newsletter”)).isSelected();
  System.out.println(“Selected status is: “+status); //true
- driver.findElement(By.name(“newsletter”)).click();
- status = driver.findElement(By.name(“newsletter”)).isSelected();
  System.out.println(“Selected status is: “+status); //false
- driver.close();

Handle Button

- ***Manual Actions***
- Check Displayed status
- Check Enabled status
- Click (Submits)
- Return the type of Element

- ***Selenium WebDriver Test Steps:***

```
System.setProperty(“webdriver.chrome.driver”, “D://chromedriver.exe”);
WebDriver driver = new ChromeDriver();
driver.manage().window().maximize();

driver.get(“http://gcreddy.com/project/admin/login.php”);

WebElement Login = driver.findElement(By.id(“tdb1”));

boolean status = Login.isDisplayed();
System.out.println(“Displaed Status is: “+status);//true

status = Login.isEnabled();
System.out.println(“Enabled Status is: “+status);//true

String elementType= Login.getAttribute(“type”);//submit
System.out.println(elementType);

Login.click();
Thread.sleep(3000);

driver.close();
```

Handle Radio Button

- ***Manual Actions:***
- Check Displayed status
- Check Enabled status
- Check Selected status
- Click (Select an Option)

- ***Selenium WebDriver Test Steps:***
- System.setProperty(“webdriver.chrome.driver”, “D://chromedriver.exe”);
  WebDriver driver = new ChromeDriver();
  driver.manage().window().maximize();
- driver.get(“https://gcreddy.com/project/create_account.php”);
- WebElement radioButton =
  driver.findElement(By.xpath(“//*[@id=\”bodyContent\”]/form/div/div[2]/table/tbody/tr[1]/td[2]/input[2]”));
- boolean status = radioButton.isDisplayed();
  System.out.println(“Displayed Status is: “+status); //true
- status = radioButton.isEnabled();
  System.out.println(“Enabled Status is: “+status); //true
- status = radioButton.isSelected();
  System.out.println(“Initial Selected Status is: “+status); //false
- radioButton.click();
- status = radioButton.isSelected();
  System.out.println(“Selected Status is: “+status); //true
- driver.findElement(By.name(“gender”)).click();
- status = radioButton.isSelected();
  System.out.println(“Selected Status is: “+status); //false
  driver.close();

Handle Dropdown box

- ***Manual Actions:***
- Check Displayed status
- Check Enabled status
- Select an Item
- Return Items Count

- ***Selenium WebDriver Test Steps:***

```
System.setProperty(“webdriver.chrome.driver”, “D://chromedriver.exe”);
WebDriver driver = new ChromeDriver();
driver.manage().window().maximize();

driver.get(“https://gcreddy.com/project/create_account.php”);

boolean status = driver.findElement(By.name(“country”)).isDisplayed();
System.out.println(“Displated Status is: “+status);//true

status = driver.findElement(By.name(“country”)).isEnabled();
System.out.println(“Enabled Status is: “+status);//true

Select dropDown = new Select (driver.findElement(By.name(“country”)));
//dropDown.selectByIndex(99);
dropDown.selectByVisibleText(“India”);

List myList = dropDown.getOptions();
int itemscount = myList.size();
System.out.println(itemscount);

Thread.sleep(4000);

driver.close();
```

WebTable:

- In Selenium, a WebTable is an HTML table that is located on a web page and is made up of rows and columns. A WebTable can be interacted with using Selenium's WebDriver API.

  There are two types of HTML tables published on the web-

Static tables: Data is static i.e. Number of rows and columns are fixed.

A static table is a table whose number of rows and columns does not change. The data within the table is also not likely to change often. You can easily locate and interact with these tables using Selenium’s WebDriver API. For ex., we can locate a specific cell in the table and retrieve its text, or we can iterate through the rows and columns of the table and perform actions on the elements within them. Selenium provides a number of methods that can be used to access and manipulate the elements within a static table.

Dynamic tables: Data is dynamic i.e. Number of rows and columns are NOT fixed.

A dynamic table is a table whose number of rows and columns can change frequently, as mentioned before. To handle the changes, you need more advanced techniques to locate and interact with the elements within a dynamic table in Selenium.

How to get rows and cols from dynamic web table:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
  public class webtableex2{
   WebDriver wd;
  public static void main(String[] args) throws ParseException{
 System.setProperty(“webdriver.chrome.driver”, “ path”);
 wd = new ChromeDriver();
 wd.get(https://money.rediff.com/gainers/bse/daily/groupa);
 int
rowsnum=driver.findElements(By.xpath(“//*[@id=‘leftcontainer’]/table/tbody/tr”)).size();
  System.out.println(“ no.of rows = “ +rowsnum);
int
colsnum=driver.findElements(By.xpath//*[@id=‘leftcontainer’]/table/tbody/tr[1]/td”)).size(
);
  System.out.println(“ no.of rows = “ +colsnum);
}}
```

Getting row data from a web table:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
 }} public class webtableex1{
   WebDriver wd;
  public static void main(String[] args) throws ParseException{
 System.setProperty(“webdriver.chrome.driver”, “ path”);
 wd = new ChromeDriver();
 wd.get(“https://money.rediff.com/gainers/bse/daily/groupa”);
 String rowdata=wd.findElement(By.xpath(“//*[@id=‘leftcontainer’]/table/tbody/tr[3]/td[4]”)).getText();
  System.out.println(“ 3rd row 4th column data = “ +rowdata);
```

Getting a column of data from webtable:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
  public class webtableex3{
 public static void main(String[] args) throws Exception{
 System.setProperty(“webdriver.chrome.driver”, “ path”);
 WebDriver  wd = new ChromeDriver();
 wd.get(“https://www.w3schools.com/html/html_tables.asp”);
 Stringbeforexpath=“//*[@id="customers"]/tbody/tr[“;
String afterxpath=“]/td[1]”;
for(int i=2;i<=7;i++){
String actualxpath=beforexpath+i+afterxpath;
webElement element = wd.findElement(By.xpath(actualxpath));
System.out.println(element.getText());
```

Handling webtable

```
Import java.util.List;
Class webtabledemo{
Public static void main(String[] args){
System.setProperty(“webdriver.chrome.driver”,”webdriver path”);
webdriver driver=new ChromeDriver();
Driver.manage().window().maximize();
Driver.get(“https://www.w3schools.com/html/html_tables.asp”);
 WebElement table=driver.findElements(By.xpath(“//*[@id="customers"]));
List<WebElement> rows=table.findElements(By.xpath(“//*[@id="customers"]/tbody/tr”));
int rowcount=rows.size();                 // gives no.of rows
List<WebElement> cols=table.findElements(By.xpath(“//*[@id="customers"]/tbody/tr[1]/th”));
int colcount=cols.size();                 // gives no.of columns
List<WebElement> cells=table.findElements(By.xpath(“//*[@id="customers"]/tbody/tr/td”));
  System.out.println(cells.size());     // prints no.of cells with data excluding header cells.
for(int i=1;i<=1;i++) {
   for(int j=1;j<=colcount;j++){
 System.out.print(table.findElement(By.xpath(“ //*[@id="customers"]/tbody/tr[“+i+”]/th[“+j+”]”)).getText()+” “);
                                                      }
           System.out.println();
          }
   for(int i=2;i<=rowcount;i++) {
      for(int j=1;j<=colcount;j++){
 System.out.print(table.findElement(By.xpath(“ //*[@id="customers"]/tbody/tr[“+i+”]/td[“+j+”]”)).getText()+” “);
                                                      }
     System.out.println();
                              }
        }
}
```

Time functions: implicit & explicit

In this topic will learn about different types of waits in Selenium:

Why Do We Need Waits In Selenium?

- Implicit Wait
- Explicit Wait
- Difference Between Implicit Wait Vs Explicit Wait

- **Why Do We Need ‘Waits’ In Selenium?**

Most of the web applications are developed using Ajax and Javascript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.

Not only it makes this difficult to identify the element but also if the element is not located it will throw an “**ElementNotVisibleException**” exception. Using Selenium Waits, we can resolve this problem.

Let’s consider a scenario where we have to use both implicit and explicit waits in our test. Assume that implicit wait time is set to 20 seconds and explicit wait time is set to 10 seconds.

Suppose we are trying to find an element which has some **“ExpectedConditions** “(Explicit Wait), If the element is not located within the time frame defined by the Explicit wait(10 Seconds), It will use the time frame defined by implicit wait(20 seconds) before throwing an “**ElementNotVisibleException**“.

Selenium Web Driver Waits

1.Implicit Wait

2.Explicit Wait

Implicit Wait in Selenium

The **Implicit Wait in Selenium** is used to tell the web driver to wait for a certain amount of time before it throws a “No Such Element Exception”. The default setting is 0. Once we set the time, the web driver will wait for the element for that time before throwing an exception.

In the below example we have declared an implicit wait with the time frame of 10 seconds. It means that if the element is not located on the web page within that time frame, it will throw an exception.

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

To declare implicit wait in Selenium WebDriver:

Implicit Wait syntax:

```
driver.manage().timeouts().implicitlyWait(TimeOut, TimeUnit.SECONDS);
```

- Implicit wait will accept 2 parameters, the first parameter will accept the time as an integer value and the second parameter will accept the time measurement in terms of SECONDS, MINUTES, MILISECOND, MICROSECONDS, NANOSECONDS, DAYS, HOURS, etc.

```
package dayone;
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;     import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;    import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
public class implicitwaitfunctiondemo{
public static void main(String[] args) throws InterruptedException
{
System.setProperty("webdriver.chrome.driver", "D:/seleniumdrivers/chromedriver-win64/chromedriver.exe");
 WebDriver driver = new ChromeDriver();
driver.manage().window().maximize();
driver.manage().deleteAllCookies();
driver.manage().timeouts().pageLoadTimeout(20,TimeUnit.SECONDS); // pageload timeout
driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS); // Implicit Wait for 15 seconds
driver.get("https://www.google.com/");
driver.findElement(By.xpath("//*[@id=\"APjFqb\"]")).sendKeys("JavaTpoint"); //Finding element and
sending values
Thread.sleep(1000);
driver.findElement(By.xpath("/html/body/div[1]/div[3]/form/div[1]/div[1]/div[4]/center/input[1]")).click
();
//Clicking on the search button if element is located
}  }
```

Explicit Wait in Selenium

- The **Explicit Wait in Selenium** is used to tell the Web Driver to wait for certain conditions (Expected Conditions) or maximum time exceeded before throwing "ElementNotVisibleException" exception. It is an intelligent kind of wait, but it can be applied only for specified elements. It gives better options than implicit wait as it waits for dynamically loaded Ajax elements.
- In order to declare explicit wait, we have to use **ExpectedConditions.** The following **Expected Conditions** can be used in Explicit Wait:

  We are using **Thread.Sleep() ,** generally it is not recommended to use

- ❑elementSelectionStateToBe()
- ❑elementToBeClickable()
- ❑elementToBeSelected()
- ❑frameToBeAvaliableAndSwitchToIt()
- ❑invisibilityOfTheElementLocated()
- ❑invisibilityOfElementWithText()

Difference between Implicit and Explicit

1. Implicit Wait applies to all the elements in the script, while Explicit Wait is applicable only for those values which are to be defined by the user.
2. Implicit Wait needs specifying "Expected Conditions" on the located element, while Explicit Wait doesn't need to be specified with this condition.
3. Implicit Wait needs time frame specification in terms of methods like element visibility, clickable element, and the elements that are to be selected. In contrast, Explicit Wait is dynamic and needs no such specifications.

Fluent Wait in Selenium

- The **Fluent Wait in Selenium** is used to define maximum time for the web driver to wait for a condition, as well as the frequency with which we want to check the condition before throwing an “ElementNotVisibleException” exception. It checks for the web element at regular intervals until the object is found or timeout happens.
- **Frequency:** Setting up a repeat cycle with the time frame to verify/check the condition at the regular interval of time
- Syntax:

```
Wait wait = new FluentWait(WebDriver reference)
.withTimeout(Duration.ofSeconds(SECONDS))
.pollingEvery(Duration.ofSeconds(SECONDS))
 .ignoring(Exception.class);
```

Page load strategy

- Page Load Strategy defines when the page would be considered loaded for the current session whenever the automation code launches a browser and uses get() or navigate().to() method.
- By default, Selenium WebDriver uses the standard Page Load Strategy, which is NORMAL. It means waiting until the entire webpage and its components, like CSS, images, frames, etc., are loaded.
- If the page takes longer time to load due to the components mentioned above and is not important to your script, you can change it to EAGER or NONE to speed up the execution.
- To determine whether the Page load is completed or not, Selenium makes use of the document.readyState property. The document.readyState describes the loading state of the document. A document here refers to any webpage that is loaded in the browser.

- The readyState property of a browser's document object represents the current state of the page loading process. The property can have one of the following values:
- **loading** – The page is still loading.
- **interactive –** The page has finished loading, but sub-resources such as images, stylesheets, and frames may still be loading.
- **complete –** The page and all sub-resources have finished loading.

Types of Page Load Strategies in Selenium

There are typically three types of Selenium Page Load Strategies that can be used in web automation.

NORMAL (Default, if not specified)

- This Selenium Page Load Strategy makes the WebDriver wait until the page load is complete, i.e., the load event is fired.
- The load event is fired when the page has loaded, including the dependent resources such as CSS, JavaScript, iFrames, and images.

Note:It waits for the HTML content to be downloaded and parsed along with all the subresources.

EAGER

- With this Selenium Page Load Strategy, the WebDriver only waits until the initial page is loaded and the DOMContentLoaded event is fired.
- Unlike the load event, the DOMContentLoaded event is fired as soon as the DOM has been loaded without waiting for additional resources like CSS, JavaScript, iFrames, and images to be loaded.

Note: It waits only for the HTML content to be downloaded and parsed.

NONE

- In this Selenium Page Load Strategy, the WebDriver is not blocked and continues with the test execution.
- It waits only for the HTML content to be downloaded.

Syntax:

```
ChromeOptions co = new ChromeOptions();
co.setPageLoadStrategy(PageLoadStrategy.NORMAL);
WebDriver driver = new ChromeDriver(co);
```