Rex Jones II

Selenium 4 New Features

Table of Contents

Video Playlist

https://www.youtube.com/playlist?list=PLfp-cJ6BH8u_4AMzeLVizVfqn4SCywSTJ

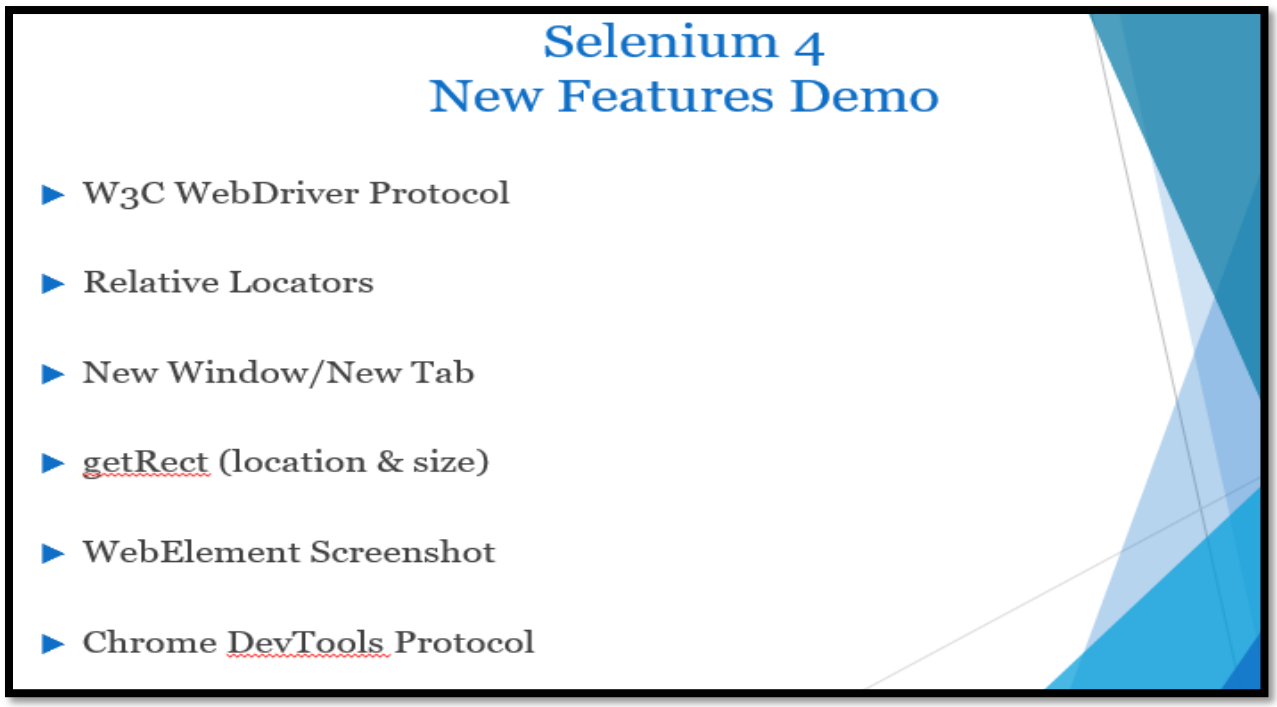
Introduction

Hello and Welcome, in this series, I am going to demo some features that have been added to Selenium 4. All components in the Selenium family have a new feature. Selenium IDE, Selenium WebDriver, and Selenium Grid.

- Selenium IDE is the component that allows us to record and playback our test
- Selenium WebDriver is an API allows us to create and execute our Test Scripts by driving a browser
- Selenium Grid is a way for us to run our test in parallel across more than 1 browser, across more than 1 operating system, and across more than 1 machine

The main reason for upgrading from Selenium 3 to Selenium 4 is the W3C WebDriver Protocol. It's the new architecture for Selenium. I will discuss new features for the complete Selenium family suite: Selenium WebDriver, Selenium IDE, and Selenium Grid.

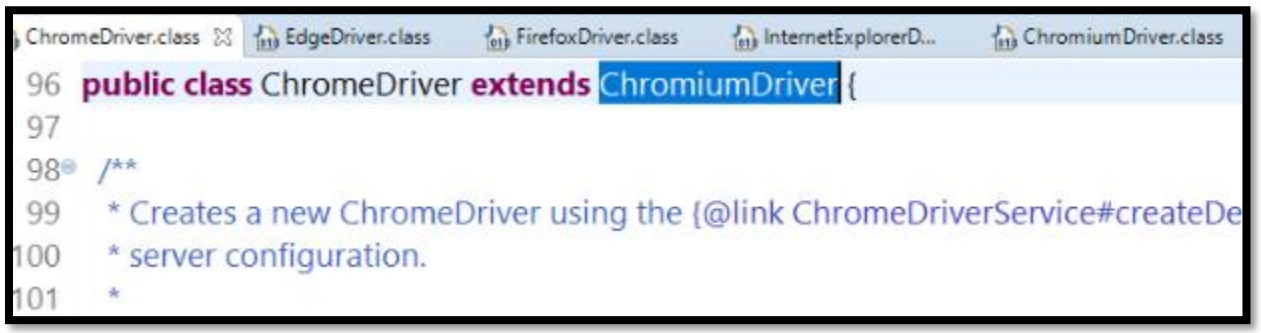
For Selenium WebDriver, I am going to explain the new Selenium 4 Architecture - W3C WebDriver Protocol then demo the Relative Locators. Relative Locators is a way to locate an element depending on the position of another element. Next, I will demo how to handle a window by switching to a new window and by switching to a new tab then take a look at getRect. getRect was implemented as a combination of getLocation and getSize. We will also take a screenshot of a WebElement. Last is the Chrome DevTools Protocol which helps us to get the Development Properties.

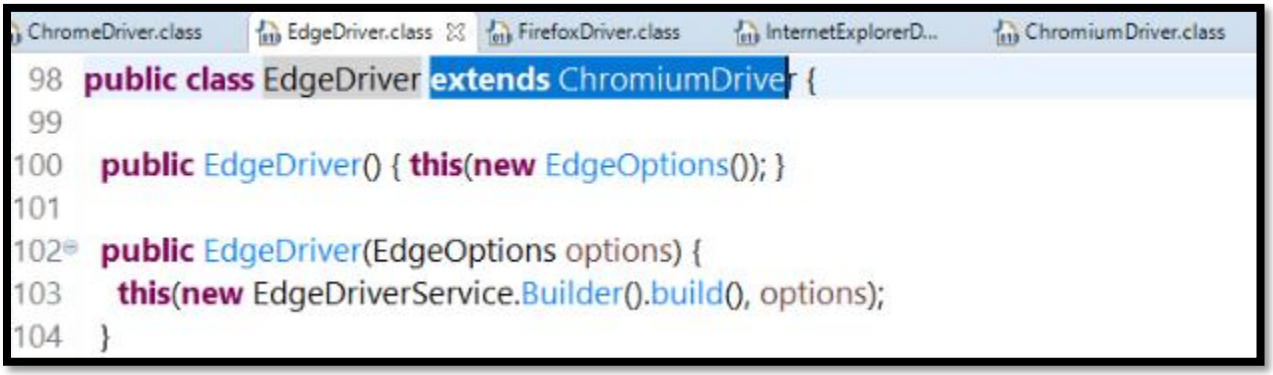


Although there are 3 components in the Selenium family suite, when we only say Selenium, we are referring to Selenium WebDriver.

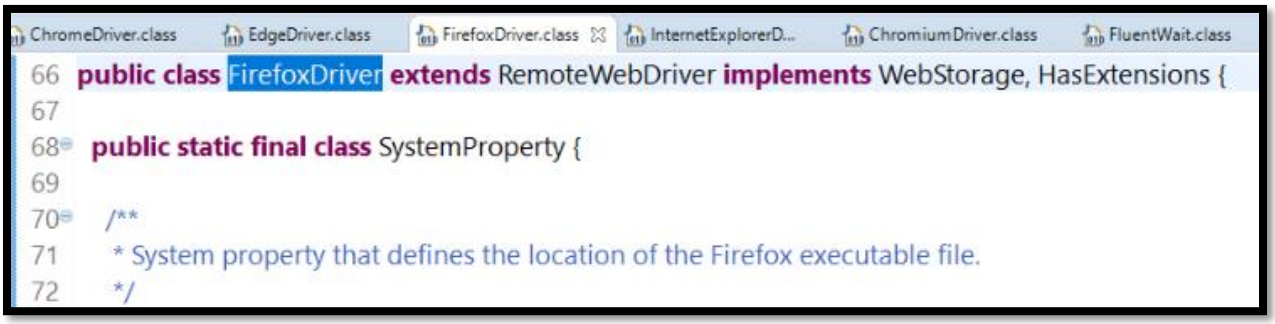
Selenium WebDriver

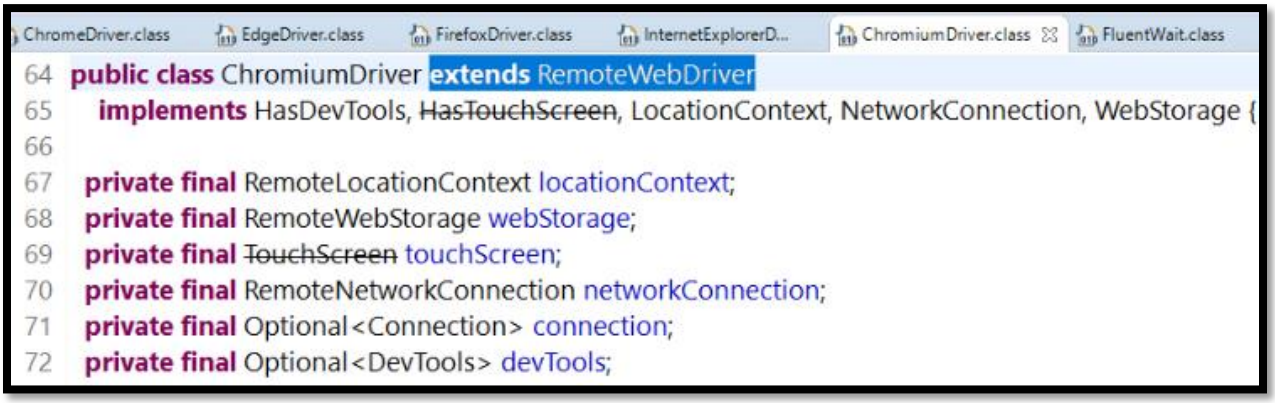
Therefore, let's start with 2 drivers that have an update: ChromeDriver and EdgeDriver. Both drivers now extend ChromiumDriver. Also, both browsers are built on the same Chromium platform.





In Selenium 3, all of the drivers including Chrome and Edge extended the RemoteWebDriver. However, with Selenium 4, FirefoxDriver, InternetExplorerDriver, OperaDriver, and SafariDriver extend RemoteWebDriver. Also, ChromiumDriver extends RemoteWebDriver. RemoteWebDriver is a way to run our test remotely but we have to configure our test.





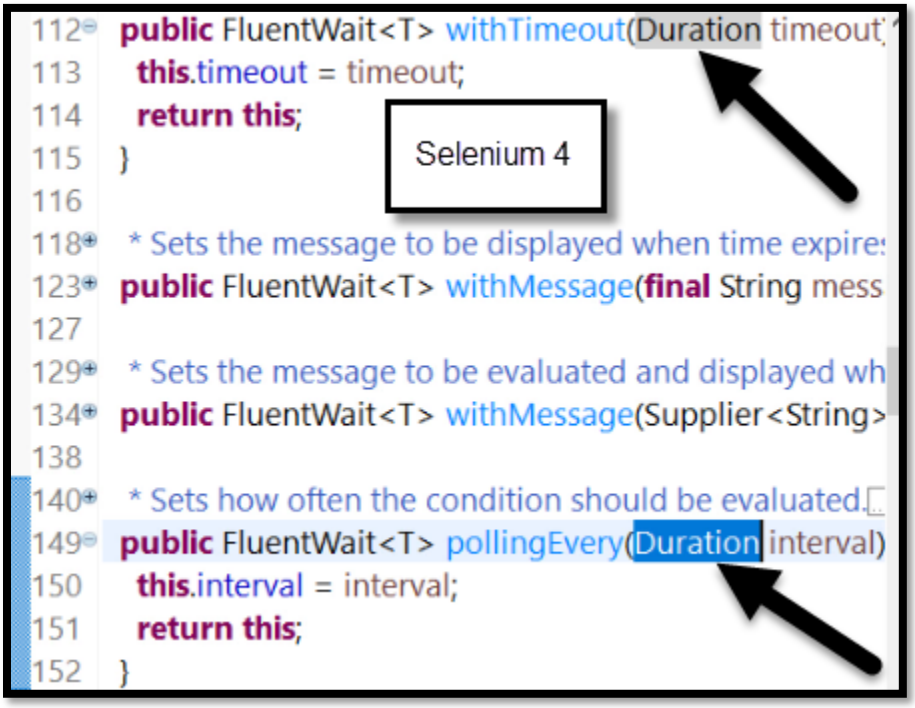
The benefit of ChromeDriver and EdgeDriver extending ChromiumDriver is access to the getDevTools method. Let me search for getDevTools and we see the DevTools class with getDevTools as the method. This method will allow us to diagnose problems and track what's going on in the browser.

```
158  @Override
159  public DevTools getDevTools() {
160    return devTools.orElseThrow(() -> new WebDriverException
161  }
162
```

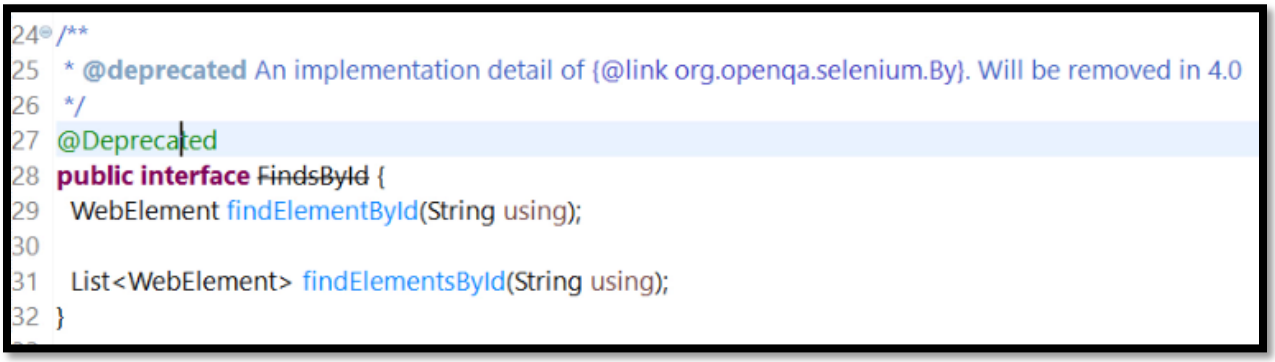
There's also a change to 2 methods in the FluentWait class. On the left is Selenium 3 and on the right is Selenium 4. The withTimeout method and pollingEvery method no longer have 2 parameters. We see long duration and TimeUnit unit but it has been replaced with only 1 parameter Duration.

```
109  public FluentWait<T> withTimeout(long duration, TimeUnit unit)
110    this.timeout = new Duration(duration, unit);
111    return this;
112  }
113
115   * Sets the message to be displayed when time expires.
120  public FluentWait<T> withMessage(final String message) {
124
126   * Sets the message to be evaluated and displayed when time expi
131  public FluentWait<T> withMessage(Supplier<String> messageSu
135
137   * Sets how often the condition should be evaluated.
147  public FluentWait<T> pollingEvery(long duration, TimeUnit unit)
148    this.interval = new Duration(duration, unit);
149    return this;
150  }
```

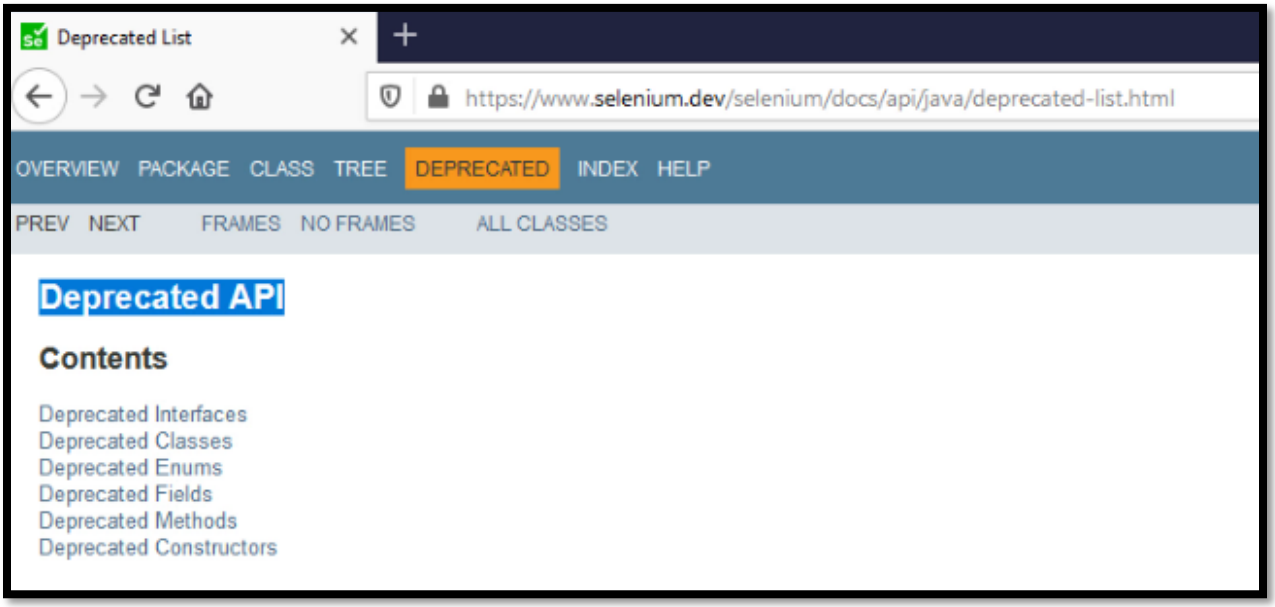
Selenium 3



Every FindsBy Interface has been deprecated and removed from Selenium 4. The other FindsBy Interfaces FindsByClassName, FindsByCssSelector, FindsByLinkText, FindsByName, FindsByTagName, and FindsByXPath have also been removed from Selenium 4.



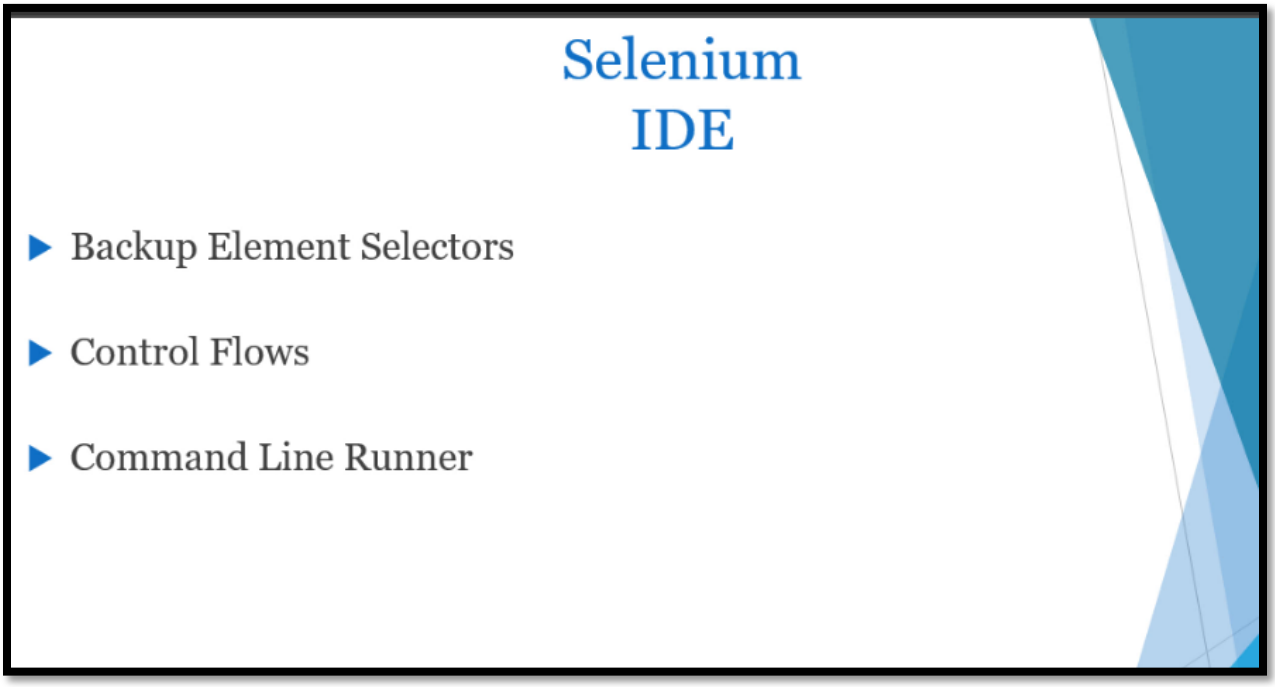
Here's a webpage that shows more deprecated API's for Selenium. If we just scroll down, we see some of the methods we already mentioned that's been removed from Selenium 4.



Next, are the new features for Selenium IDE.

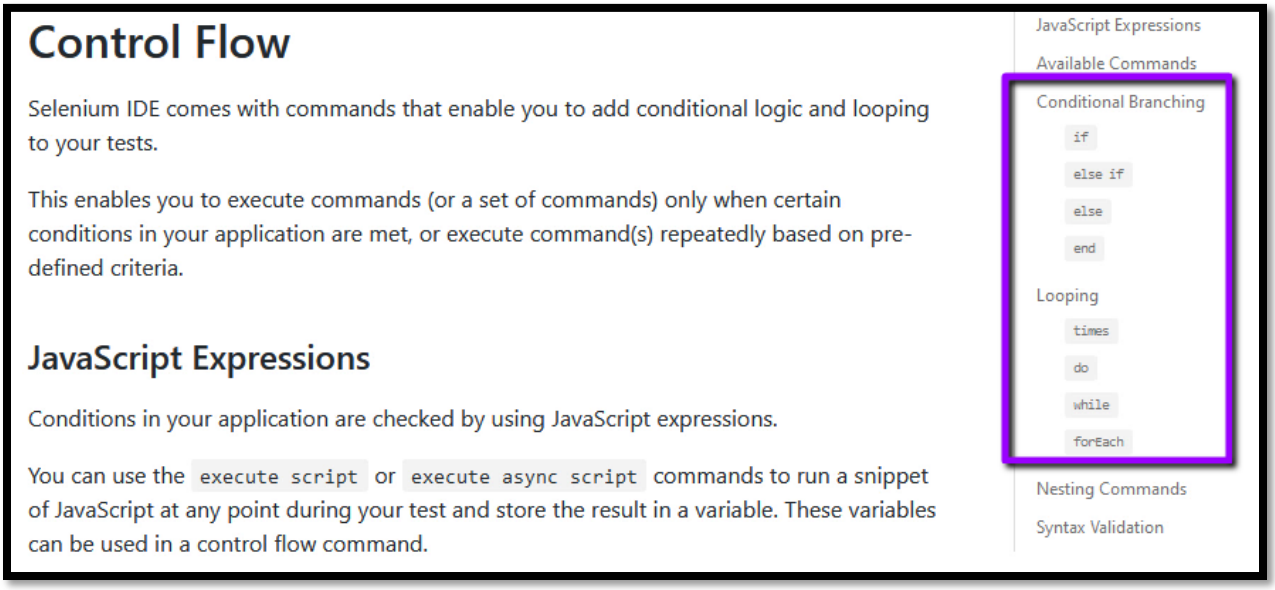
Selenium IDE

When it comes to Selenium IDE, the purpose is to record actions we take on a website then playback those actions from the website. In the past, Selenium IDE was only available as a Firefox extension but now it's available as a Firefox extension and Chrome extension. Microsoft Edge will be available soon as an extension. There are plans for Selenium IDE to be available as an Electron app. The Electron app will allow us to use the Debugging Protocol and listen out for events from the browser. Some new features include the Backup Element Selectors, Control Flows, and the Command Line Runner also known as CLI runner.

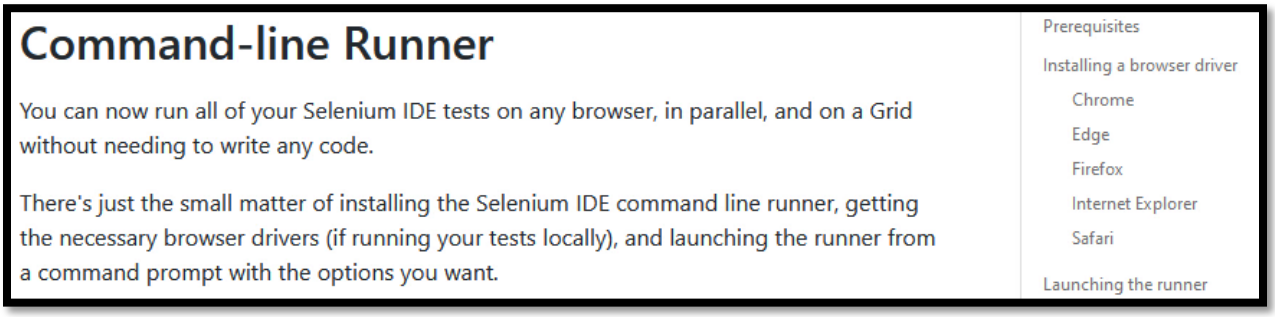


The Backup Element Selectors record more than 1 locator for each element. If we execute a test that does not locate an element then this feature will fall back to a different locator until the element is found. For example, our test runs and it cannot locate an element using a linkText locator. The Backup Element Selectors will check to see what other locators were recorded for this element then grab another locator such as id, xpath, or cssSelector.

There are 2 types of Control Flows. We have Conditional Branching and Looping. The Conditional Branching Flow decides what happens next if a certain condition is satisfied or not satisfied. Loop Control Flows execute an instruction for a number of times. We see the commands for each flow are if, else if, else, and end for Conditional Branching while Looping has times, do, while, and forEach.



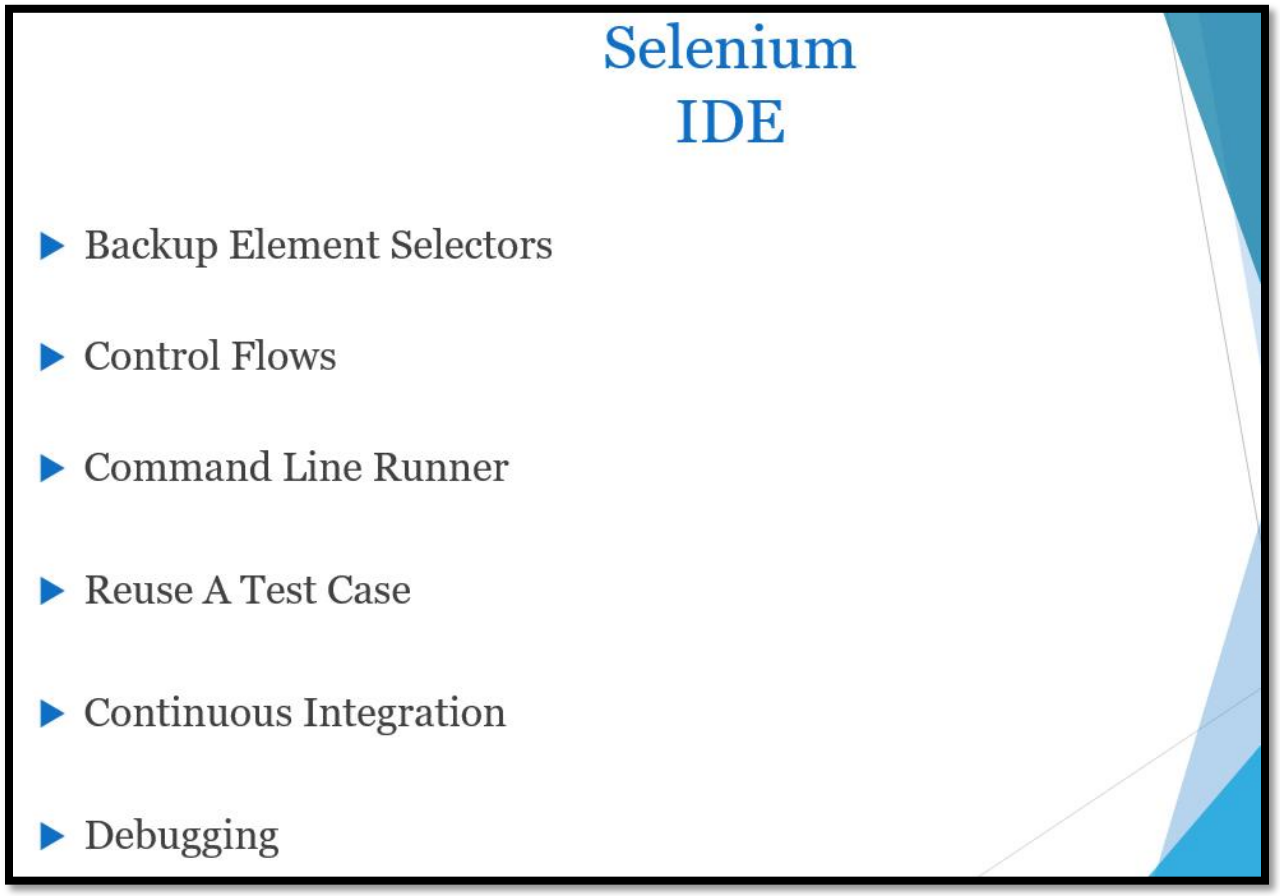
The Command Line Runner also called Selenium side runner allows us to run our test on any browser: Chrome, Edge, Firefox, Internet Explorer, and Safari. In this 1st sentence, it says "in parallel, and on a Grid without needing to write any code". That's why we can imagine Selenium IDE as an on ramp to the Selenium family because little to no programming knowledge is required to start automating our test. By the way, the new name is called Selenium IDE TNG. TNG is an acronym for The Next Generation so the complete name is Selenium IDE The Next Generation.



Do you see Code Export under Introduction? Code Export is not a new feature but it's a good feature that pushes code to Selenium WebDriver in your chosen language and test framework. To export, all we do is right click a test or suite, select Export then pick a supported language, test framework and click the Export button. Oh, I almost forgot, Selenium IDE has a way to create new commands as part of their plugin system. Those plugins can be delivered as an extension. A big shout out goes to Applitools because they are the ones who revived Selenium IDE and has a plugin that makes it possible to perform codeless visual testing.

We can also reuse a Test Case implementing a run command and a store command. The run command executes a test from within another test. Store is a command that allows us to save a value that we can

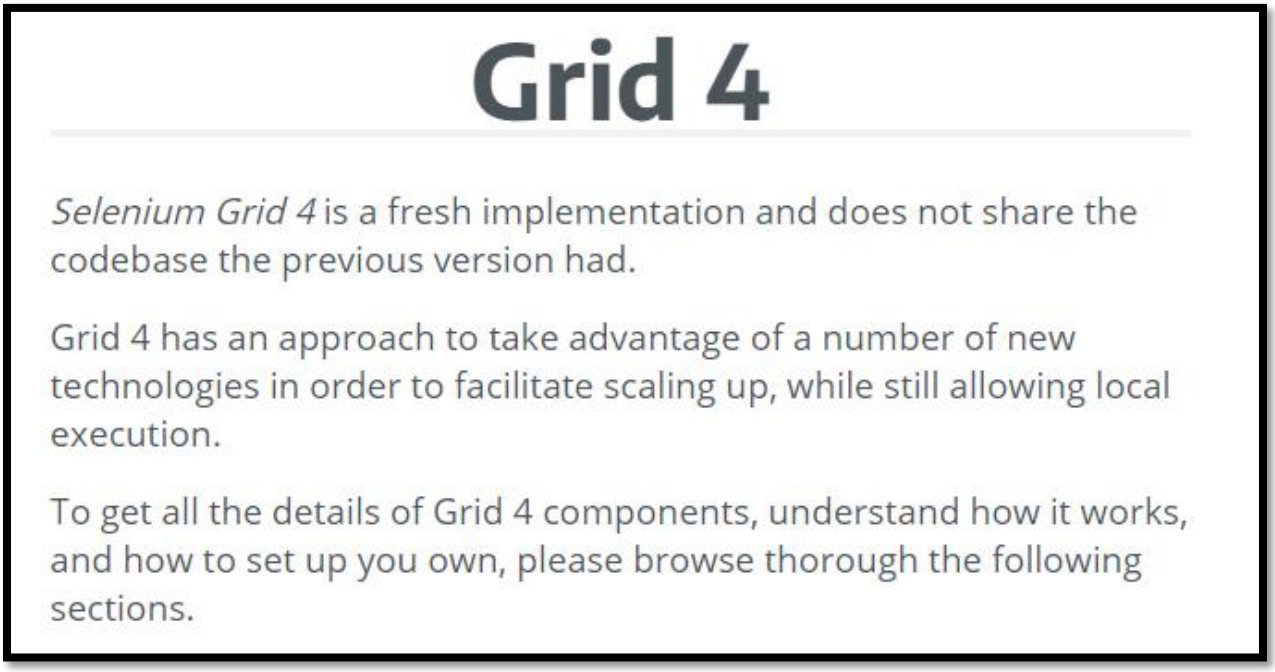
reuse later. Last but not least is Continuous Integration and Debugging. Selenium Side Runner assist with integrating our code continuously into a shared repository. The debugging feature allows us to set breakpoints to pause our test at a certain point, play our test to a certain point, record our test from a certain point, and play our test from a certain point.



Next, let's talk about the new features for Selenium Grid.

Selenium Grid

The new feature for Selenium Grid involves a brand-new architecture. On this selenium.dev page, we see Grid 4 has an approach to take advantage of a number of new technologies. You can browse these sections for more information about the components, understand how the Grid works, and set up the Grid. Let's discuss the components and how the grid works.

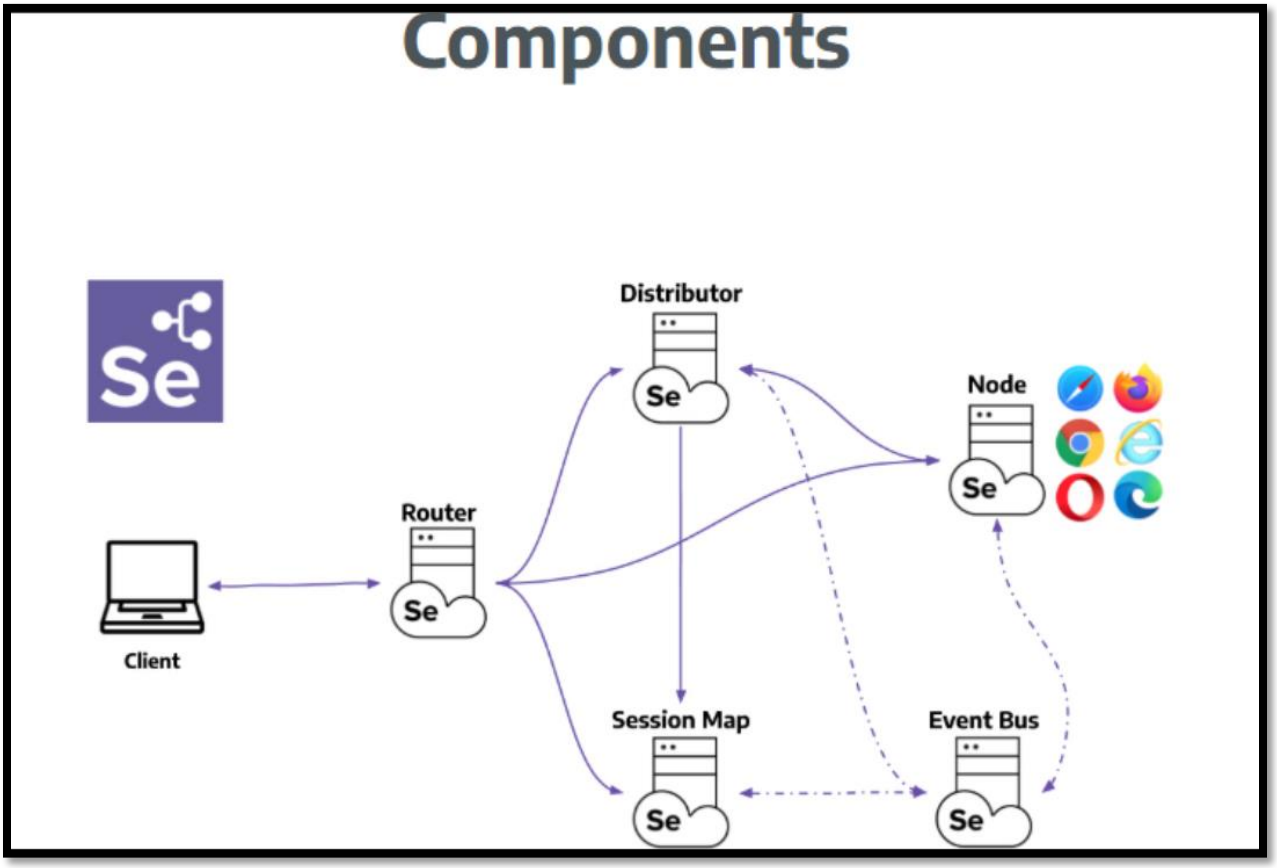


We have a Client, Router, Distributor, Session Map, Node, and Event Bus. Also, GraphQL has been added as a new way query and get data. I'm going to get straight to the point and discuss these Components. The Router's role is to listen for a session request. We see the Router has an arrow pointing to the Distributor and Session Map.

If it is a new session request then communication starts between the Router and Distributor. If the session request already exists then communication is between the Router and Session Map.

When it comes to the Router and Distributor, a session request is sent to the Distributor. The distributor is responsible for selecting a node to run our test. This diagram shows 1 node but in reality, we set up more than 1 node. The node can be a physical machine or virtual machine that execute our Test Scripts. After the distributor selects a node to run our test, the node responds back with a URL of the session. At this point, a session is created and the Distributor sends information to the Session Map. The Session Map is important because it maps the session ID.

That's how the Grid works when there's a new session request. When the session already exists, the Router sends a Session ID to the Session Map. In return, the Session Map sends the node that has a running session back to the Router. For subsequent steps, the Event Bus operates as a path for communication to other Grid components.



In addition to the new architecture, Selenium Grid is more Modern and has Observability. By it being more modern, we can leverage up-to-date technologies like Docker and Kubernetes. With Docker, our applications can run in containers and not worry about setting up virtual machines. Kubernetes allows us to scale using a cloud infrastructure.

Observability helps us to trace, log, and measure what's going on with the system internal state. There will be a trace id when a request comes in to help developers and admins debug a problem. That's it for the new features for Selenium Grid 4 and I will see you in the next session to discuss the new W3C WebDriver Protocol for Selenium 4 WebDriver.

W3C WebDriver Protocol

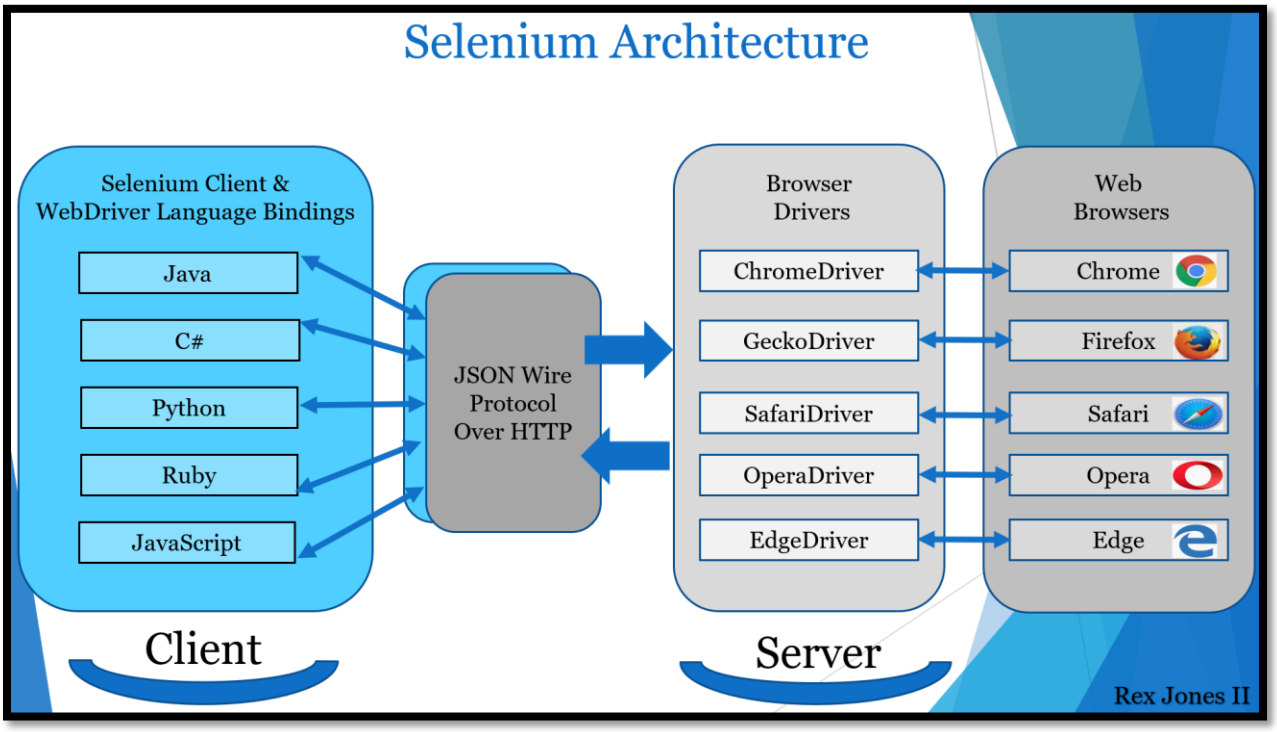
The W3C WebDriver Protocol has at least 3 advantages. #1 It provides standards #2 It provides stability and #3 It provides an updated Actions API that is supplied with better resources. I will talk about all 3 and get straight to the point.

If you are interested in more videos, you can subscribe to my YouTube channel and click the bell icon. You can also follow me on Twitter, connect with me on LinkedIn and Facebook. I will also place the transcript and presentation slides on GitHub.

In this session, I am going to speak about the Selenium 4 Components, Advantages, Mobile Automation, and Cloud Platforms.

Components

Starting with the components, Selenium has moved from 3 to 4 because of the W3C WebDriver Protocol. This is an example of Selenium 3 which includes the JSON Wire Protocol. The objective of JSON Wire Protocol was to transfer information from the client to the server. That information was processed over HTTP by sending HTTP Requests and receiving HTTP Responses.

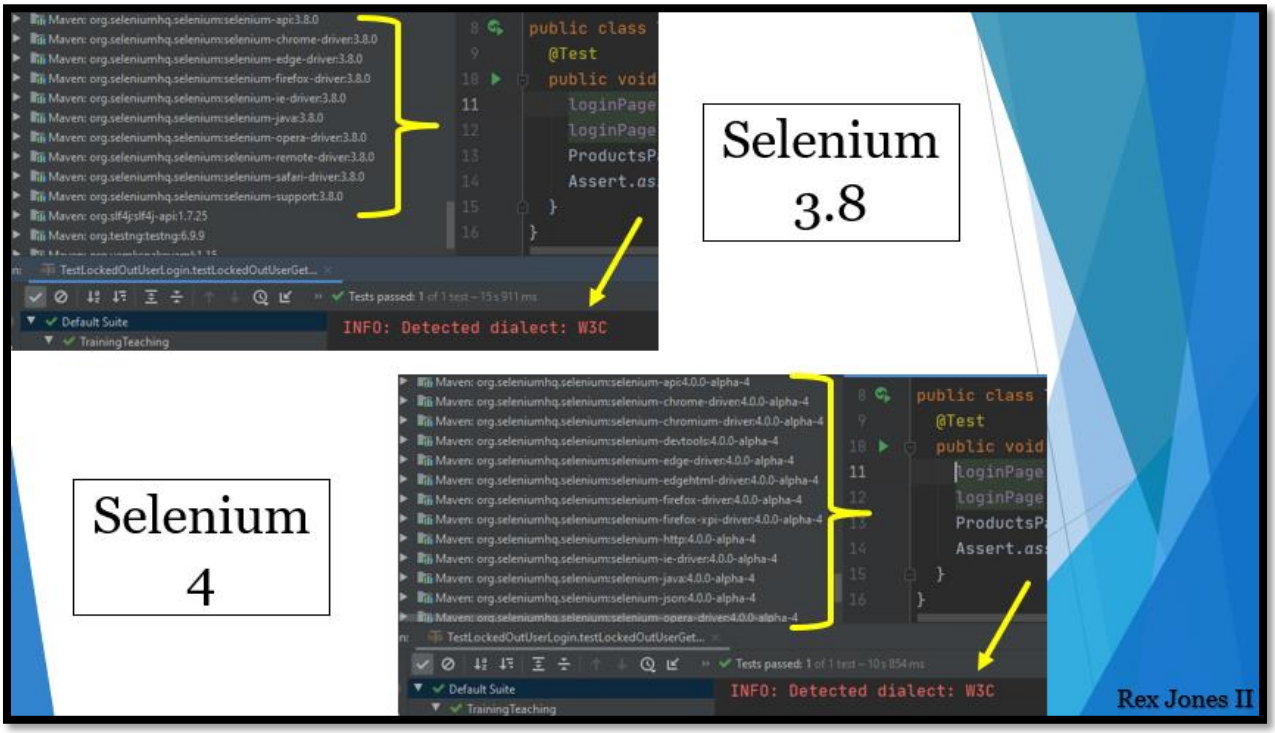


Direct

With Selenium 4, the JSON Wire Protocol has been removed from the new architecture. Now, there is direct communication between the Browser Drivers and Selenium Client & WebDriver Language Bindings. The 1st component has 2 parts combined into one. Selenium Client is a separate part and

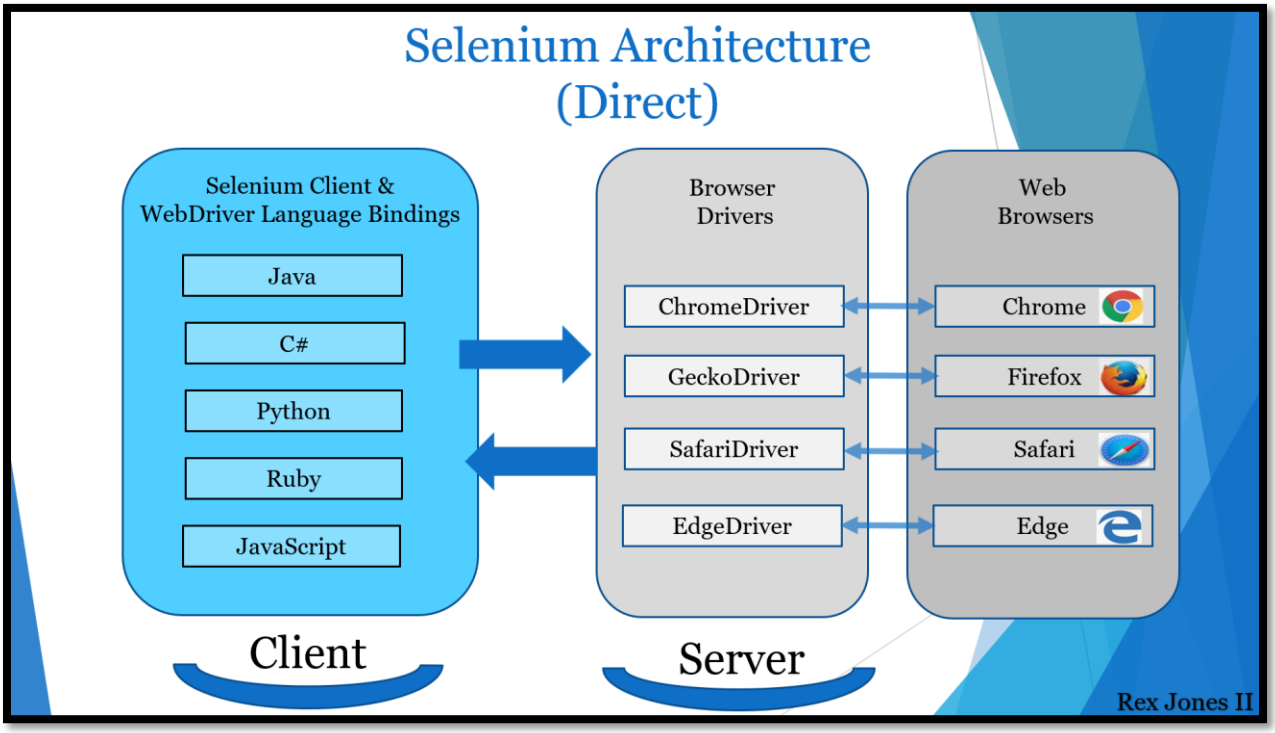
WebDriver Language Bindings is a different part. Selenium is an API that have commands for automating our browser. WebDriver has only 1 job and that job is talk to the browser through a driver.

Each language has their own bindings. Bindings mean the same commands written for Java is also written for C#, Python, Ruby, and JavaScript. You may have noticed that Selenium added support for the W3C protocol starting at version 3.8. According to Simon Stewart, in this Selenium 4 Webinar with BrowserStack, he mentioned the versions of Selenium since 3.8 have spoken to both JSON Wire Protocol and W3C Protocol. After running your Test Script, look for INFO: Detected dialect: W3C to see if your Selenium version is speaking to W3C. Both of these screenshots show W3C for Selenium 3.8 and Selenium 4. If not W3C then it will show OSS which means Open Source Software.



Back to our diagram, we see the 2nd component is Browser Drivers and it have 2 functions. The first function is to receive a request from Selenium Client & WebDriver Language Bindings then pass that request to the browser. A driver also known as a proxy is responsible for controlling the browser. The second function is to return a response from the browser back to the Selenium Client & WebDriver Language Bindings. All of the drivers use the W3C WebDriver Protocol and most of them are created by the browser vendors.

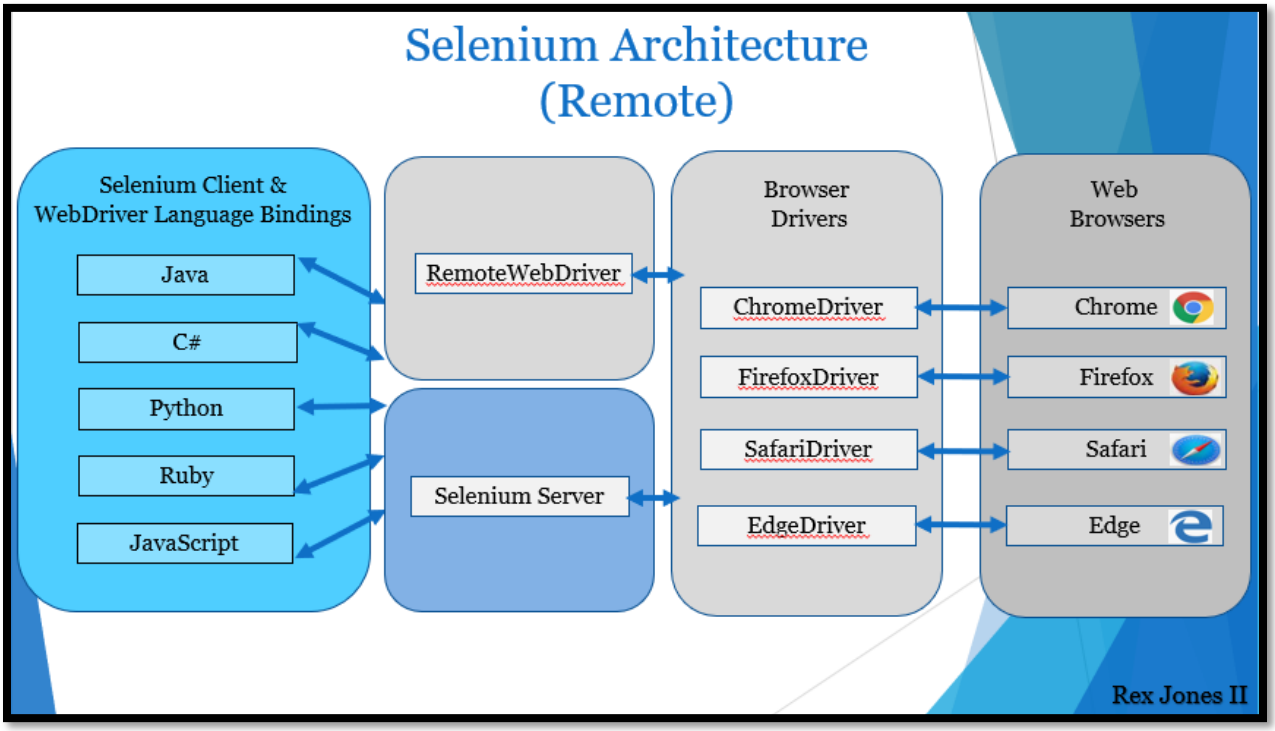
When it comes to the 3rd component Web Browsers. This is where all of the Selenium Commands are performed. The browser receives a request, performs the request, and sends back a response to the driver. Also, notice how Opera is not available as a driver or browser. In reality, they are still available but I did not add them because the WebDriver implementations are no longer under development so native support has been removed for Opera and Phantom JS.



Remote

Remote is another form of communication to the browser. It can happen through the RemoteWebDriver or Selenium Server. The RemoteWebDriver is a class that implements the WebDriver Interface. With Selenium 4, FirefoxDriver and SafariDriver continue to extend RemoteWebDriver. However, ChromeDriver and EdgeDriver no longer extend RemoteWebDriver but they extend ChromiumDriver. In the introduction, I showed how ChromiumDriver, FirefoxDriver, InternetExplorerDriver, OperaDriver, and SafariDriver all extend RemoteWebDriver. We see 3 of the boxes (RemoteWebDriver, Browser Drivers, and Web Browsers) are gray because they all run on the same system.

The Selenium Server is different. It's a way to communicate remotely when talking to the driver but not on the same system as the driver. That's why Selenium Server is a different color. We start the server by using the Selenium Standalone jar file. After it starts, the server directs our Test Scripts to a remote web browser.



Advantages

Standards

For W3C WebDriver Protocol, the advantages are Standards, Stability, and Actions. W3C stands for World Wide Web Consortium which is an international group of people that create long term standards for the web. With that 1st advantage of standards, our Test Scripts run consistently on each browser. There were times with Selenium 3 that some commands performed offbeat on different browsers. Since Selenium 4 is compliant with W3C WebDriver, there is no more required encoding and decoding of the API request.

Stability

The second advantage is stability. I believe backward compatibility is the main benefit of stability. Per Simon Stewart, they are fully aware that some people will want to use the old JSON Wire Protocol. So, the Java Bindings and the Selenium Server will provide mechanisms for people to use the old JSON Wire Protocol. They know, we have spent time, we have spent effort, and we have been dedicated to building up our Test Suite. Therefore, our Test Suite will remain smooth for Selenium 4. No changes to our Test Scripts unless an API has been marked deprecated. Deprecated API's like the FindsBy Interfaces have been removed from Selenium 4. As a result, the WebDriver API's are going to continue working like there was never a change to Selenium.

Actions API

The updated Actions API is the third advantage. With this API, we can handle keyboard and mouse events like double clicking an element. It's an advantage because Selenium 4 offers a way to perform more than 1 action at the same time like pressing 2 keys. That's an advantage for UI Automation.

Mobile Automation

For Mobile Automation, Appium is the tool used for automating mobile applications. We see on Appium.io site, the introduction shows native, mobile web, and hybrid application on iOS mobile, Android mobile, and Windows desktop platforms. The Appium client library have implemented elements of the W3C Protocol.

Cloud Platforms

Also, Cloud Platforms such as SauceLabs and BrowserStack support W3C WebDriver. However, the format of their capabilities will change. You can go to their site to see what capabilities need to be updated. Here's the page for BrowserStack with an Introduction section, talk about Why changes are being made, What does it mean to me, Updating the Selenium tests, and an Example section that show how capabilities are passed with W3C Protocol. SauceLabs have a page that shows W3C Capabilities Support with sections: What You Will Need, Verifying the Capabilities, W3C WebDriver-Compliant, Instantiating WebDriver with W3C, and Common Test Script Configuration Errors To Avoid. That's it for the W3C WebDriver Protocol. Don't forget to Connect and Subscribe. Next, I will demo the Relative Locators which locate elements based on their relationship to another element.

Relative Locators

Selenium 4 provides 5 Relative Locators to find an element or elements located above(), below(), near(), toLeftOf(), and toRightOf() another element.

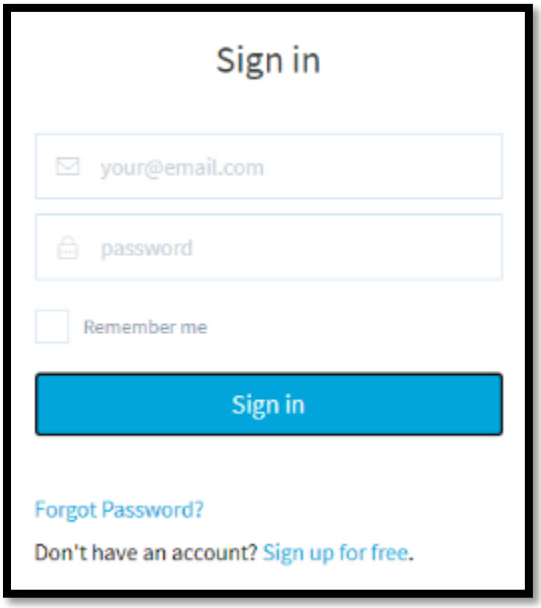
In this session, we will use TestProject's website. It's a free E2E test automation platform for web, mobile, and API. If you are interested in more videos, you can subscribe to my YouTube channel and click the bell icon. Also connect with me on LinkedIn, Facebook, and follow me on Twitter. Lately, I have been slow creating my transcripts but I will make sure to place them on GitHub.

Okay, let's go ahead and get straight to the point. Relative Locators were formerly called Friendly Locators. The TestNG Configuration methods BeforeMethod and AfterMethod are pre-written. BeforeMethod will set up our test and AfterMethod will tear down our test.

```
WebDriver driver;

@BeforeMethod
public void setUp () {
  WebDriverManager.chromedriver().setup();
  driver = new ChromeDriver();
  driver.manage().timeouts().implicitlyWait( time: 5, TimeUnit.SECONDS);
  driver.manage().window().maximize();
}
```

```
@AfterMethod
public void tearDown () {
  driver.quit();
}
```

Our Test Script is testRelativeLocator_Below. It will find the Sign In button then click the Forgot Password link below the Sign In button.



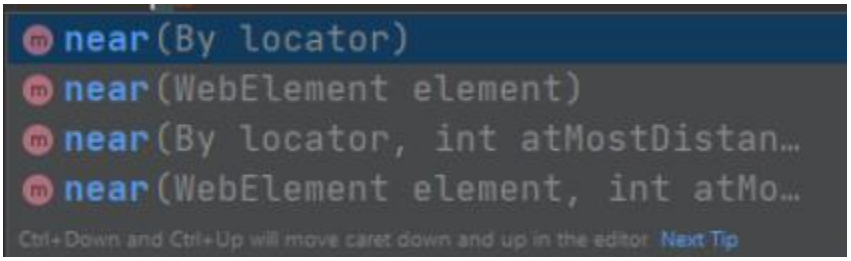
Let's inspect the Sign In button and we see the value for id is tp-sign-in. Inspect the Forgot Password link and we see the value for the Tag Name is 'a' is the tagName. 'a' is for hyperlink. That's all we need to click the Forgot Password link.

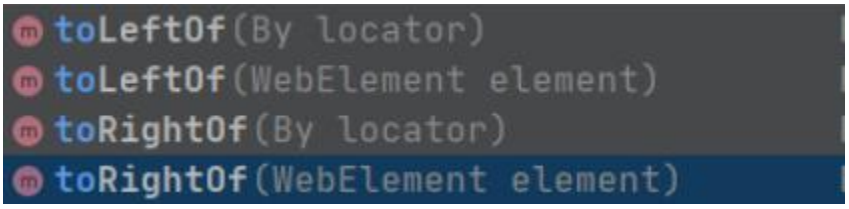
(1) Below (Relative Locator)

There are 2 ways to write a Relative Locator. We start by writing driver.findElement(RelativeLocator.withTagName) and the other way is to bypass writing RelativeLocator and to start by using withTagName. Import.

However, there's a difference with the import statements. To use withTagName without writing Relative Locator, we import the package with static. If you want to use the other way with Relative Locator.withTagName. It's the other package without static. By default, Eclipse does not import the static package.

The tagName for Forgot Password is "a" so we enter "a" dot. Here's a list of our Relative Locators. I want you to notice something. Each Relative Locator is overloaded with more than 1 option: above has a By locator parameter and a WebElement element parameter, the same with below: By locator, WebElement element, near is the only locator with 4 options (By locator, WebElement element, By locator, int atMostDistanceInPixels, and WebElement element), int atMostDistanceInPixels), toLeftOf has a (By locator and WebElement element_, the same with toRightOf (By locator and WebElement element).





Right now, we are trying to find the Forgot Password link which has a tagName of "a" and located below the Sign In button so we select below with a By locator parameter. We write the locator for the Sign In button: By.id("tp-sign-in")). Now, we can also do 1 more thing after entering the value for id. When it comes to a link Forgot Password, next is to click the Password link. If you wanted to, you can also use WebElement by writing WebElement signInButton = driver.findElement(By.id("tp-sign-in")). Next, we pass in signInButton. Either way is okay. Now we have the Forgot Password link and we are clicking the Password link. Next, let's go ahead and see if we actually land on the next page. Go back to the AUT and click the Forgot Password link and to verify we landed on the Forgot Password. We are going to inspect

Reset Password and it has an id value of tp-title. If we wanted to, we can also use an assert statement but I want to make sure a statement is printed to the Console. Relative Locators can be good for dynamic elements. Dynamic elements are elements like tp-title that changes every time but in this case the value is not dynamic. If the value changes but the tagName remains the same then that's a good reason to use Relative Locator.

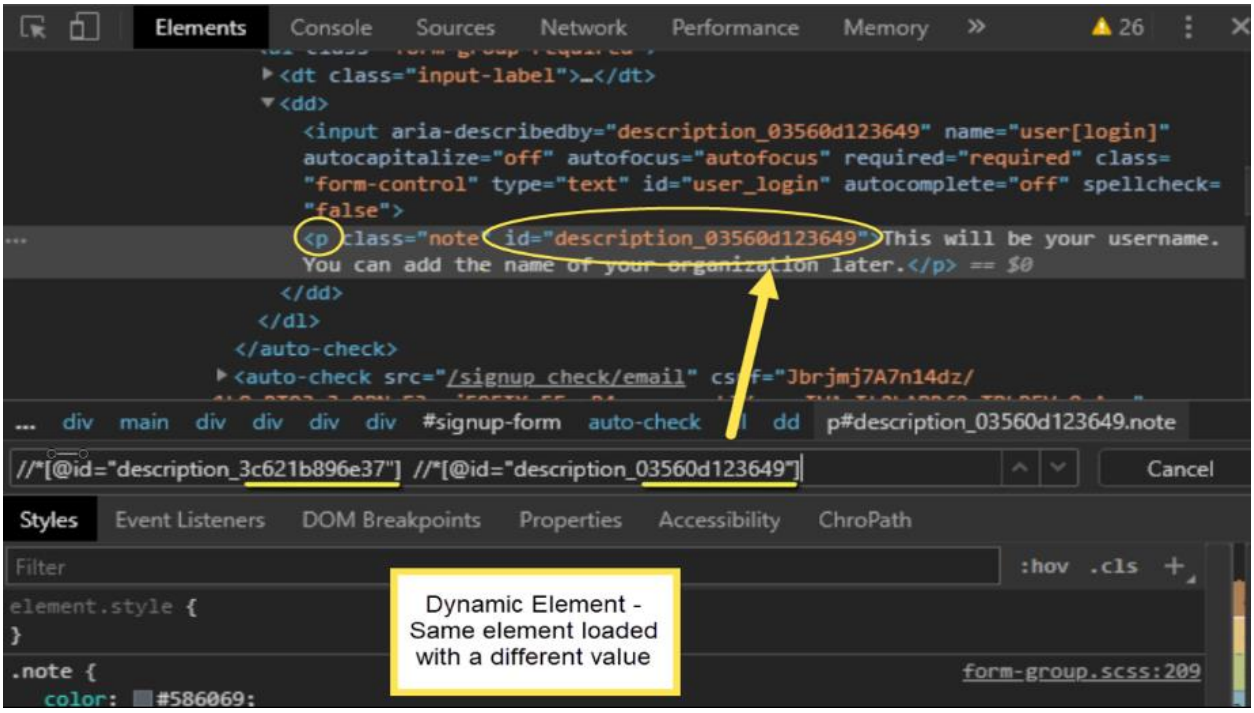
Write String title = driver.findElement(By.id("tp-title")).getText());
sysout("Title: " + title)

```
@Test
public void testRelativeLocator_Below () {
  driver.get("https://app.testproject.io/");
  WebElement signInButton = driver.findElement(By.id("tp-sign-in"));

  driver.findElement(withTagName("a").below(signInButton)).click();
  String title = driver.findElement(By.id("tp-title")).getText();
  System.out.println("Title: " + title);

}
```

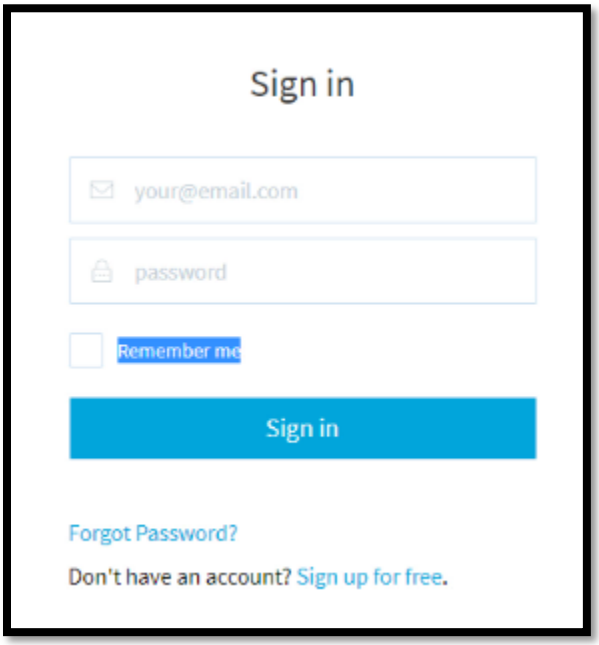
Let's Run. Bingo Title = Reset Password.

(2.1) Above-ToRightOf (Relative Locators)

In this session, we will look at the benefit of using more than 1 Relative Locator. I mentioned in the previous session that Relative Locators are good for Dynamic Elements. Here's an example of a dynamic element that I covered in my xpath video 87 talking about "Why We Should Never Copy XPath Values". There are 2 id values. On the left, the value ends with 37 and on the right, the value ends with 49. In this screenshot, the arrow is pointing to the most recent id value that ends with 49. The value changed but the tagName remained at p for paragraph.



We can include Relative Locators to the other locators such as xpath and cssSelector for finding Dynamic values. For this session, let's use the same Sign In page as the previous session but this time get the text of Remember me which is above the Sign In button and to the right of the checkbox.



Let's inspect the checkbox and the id value is rememberMe. We also need the tagName of the Remember me verbiage. So, let's it Inspect and the tagName is span. We are going back to IntelliJ and I

need to copy the element for the Sign In button and paste in our new Test Script. We also need to go ahead and get the value of the Remember Me checkbox so I'm going to write WebElement rememberMeCheckbox = driver.findElement(By.id("rememberMe"));

Now, let's find the Remember Me text by writing WebElement rememberMeText = driver.findElement(withTagName("span"))
The verbiage is located .above(signInButton));

Print the verbiage sout("Text = " + rememberMeText.getText());

```
@Test
public void testMultipleRelativeLocators_AboveToRightOf () {
  driver.get("https://app.testproject.io/");
  WebElement signInButton = driver.findElement(By.id("tp-sign-in"));
  WebElement rememberMeCheckbox = driver.findElement(By.id("rememberMe"));
  WebElement rememberMeText = driver.findElement(withTagName("span")
          .above(signInButton));
  System.out.println("Text = " + rememberMeText.getText());
}
```

Let's run. We see Text is blank. Why is it blank? We see is blank. Why is it blank? It is blank because the checkbox and text are both located above the Sign In button. Our Test Script found the checkbox. Inspect the checkbox again and we see it also has a span tag just like the Remember me. The checkbox was selected for the Relative Locator because it's the first span tag.

Recall, the findElement method finds the first WebElement so that's why checkbox was selected. In this scenario, we need to use multiple Relative Locators because it's more accurate than only using 1 Relative Locator. Remove the semi-colon and add the dot. The text was located above the checkbox and .toRightOf(rememberMeCheckbox);.

```
@Test
public void testMultipleRelativeLocators_AboveToRightOf () {
  driver.get("https://app.testproject.io/");
  WebElement signInButton = driver.findElement(By.id("tp-sign-in"));
  WebElement rememberMeCheckbox = driver.findElement(By.id("rememberMe"));
  WebElement rememberMeText = driver.findElement(withTagName("span")
          .above(signInButton)
          .toRightOf(rememberMeCheckbox));
  System.out.println("Text = " + rememberMeText.getText());
}
```

Now. let's run again and see what happens. Now, text has a value equal to Remember Me and it PASSED. Next, let's find a list of WebElements.

(2.2) ToRightOf (Relative Locator) / Find List Of Elements

Let's get the URL for our next Test Script. I'm going to copy https://addons.testproject.io and paste it to take us to the Addons page. We are going to get the names of each platform. If I inspect each icon, the tagName is img and the names are located in attribute alt. We see Platform Web, Platform Android, and the third icon is Platform iOS. They are located to the right of this textbox. Inspect the textbox and the id value is q.

To find a list of elements using a Relative Locator, we start by writing List then <WebElement> and the name will be allPlatforms = driver.findElements(). Find elements with an 's' find all of the elements (withTagName("img"))
to the right of text box .toRightOf(By.id("q")));

Here's the logic, we must loop through each image then print the name.

for (WebElement platform : allPlatforms) {)
sout(platform.getAttribute("alt"));

```
@Test
public void testRelativeLocator_FindListOfWebElements () {
  driver.get("https://addons.testproject.io/");
  List<WebElement> allPlatforms = driver.findElements(withTagName("img")
          .toRightOf(By.id("q")));

  for (WebElement platform : allPlatforms) {
    System.out.println(platform.getAttribute( name: "alt"));
  }
```

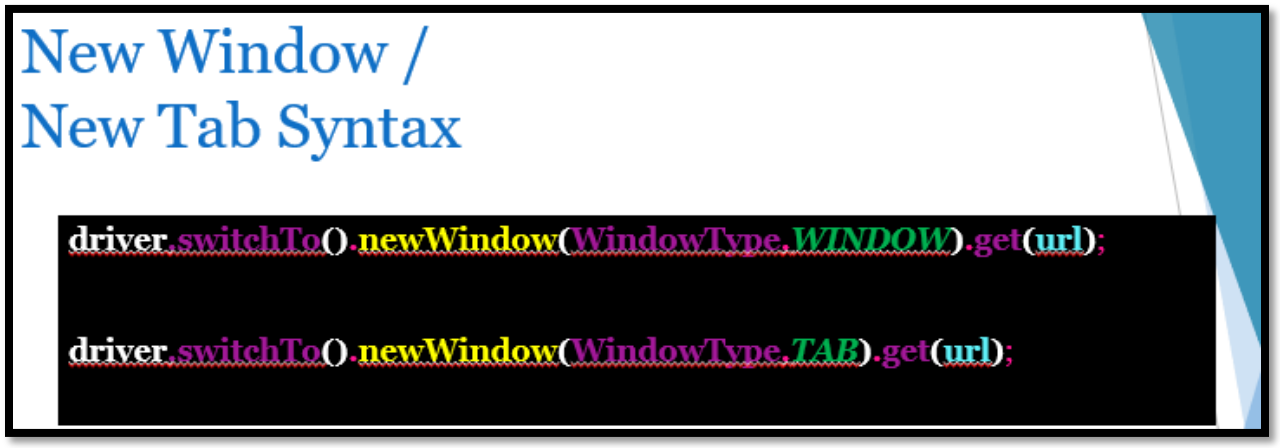
Remember the names were located in the alt attribute so that's why I wrote getAttribute and not getText. I created a video 12 that explains the difference between getAttribute and getText if you want to check it out. Let's Run. Bingo, we see all of the platform names: Platform Web, Platform Android, Platform iOS and they PASSED. That's it for using more than 1 Relative Locator to find a WebElement and how to use a Relative Locator to find a list of WebElements. I have more videos on the way. You can like, subscribe and click the bell icon. Plus connect with me on LinkedIn, Twitter and Facebook.

New Window/New Tab

Selenium 4 allows us to open a new window and open a new tab in the same session at the same time. After opening the window or tab, we have the ability to work in it without creating a new driver.

Concept of Opening A New Window/Tab

In this video, we are going to look at the concept of opening a new window and a new tab. With previous versions of Selenium, we could use the Robot Class to simulate the keyboard and open a new tab by pressing CTRL + T then switching focus to the tab after getting the window handle. Now, we can perform those same steps with Selenium 4 using 1 line of code: We can use it for a window or tab.
driver.switchTo().newWindow(WindowType.WINDOW or TAB).get(url);



Notice, WINDOW and TAB are written in green.

- driver is used to control the browser
- switchTo() is a method that sends commands to a window. It switches focus between the windows
- newWindow() is a method that creates a new window or tab then automatically switches focus to that new window or tab
- WindowType is an enum. Enum is short for enumeration which is a list of constants that define a new data type. In this case, the constants are WINDOW and TAB that instructs our program to open a new window or new tab.
- WINDOW is the parameter for creating a new window
- TAB is also a parameter but it's used for creating a new tab
- Last is the get(url) method which loads a new page in our browser

New Window / New Tab Syntax

```
driver.switchTo().newWindow(WindowType.WINDOW).get(url);

driver.switchTo().newWindow(WindowType.TAB).get(url);
```

Argument	Description
driver	Controls the browser
switchTo()	Sends commands to a window
newWindow()	Create a new browser window or tab then automatically switch focus
WindowType	An enum containing a list of constants (WINDOW and TAB)
WINDOW	A type hint parameter that creates a new window
TAB	A type hint parameter that creates a new tab
get(url)	Loads a new page in the browser

For our Test Script, we have a set up method to load an Automation Practice website then get the title. I commented out the tear down method because I want the page to stay open after executing our Test Script.

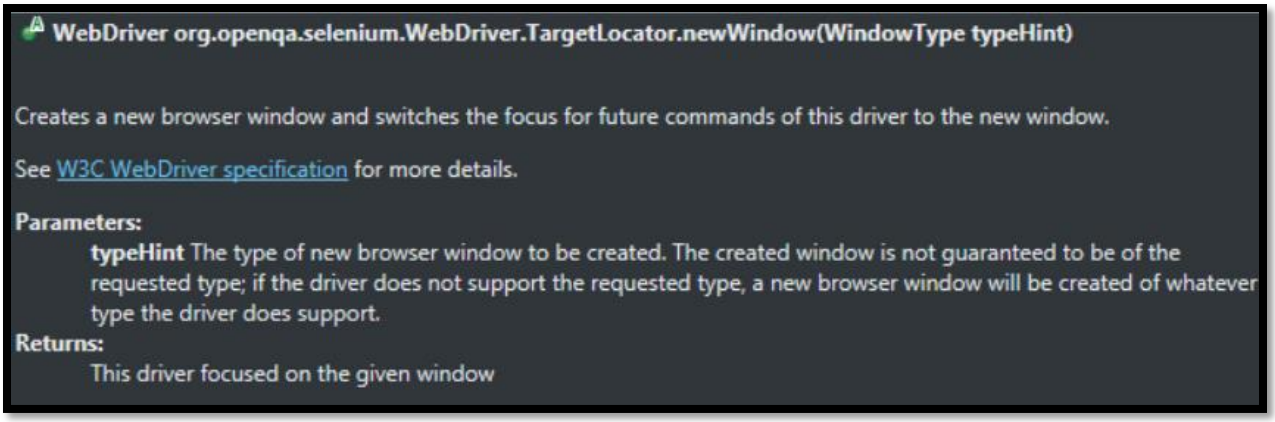
```
@BeforeMethod
public void setUp () {
  WebDriverManager.chromedriver().setup();
  driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.get("http://automationpractice.com/index.php");
  System.out.println("Title: " + driver.getTitle());
}
```

```
@AfterMethod
public void tearDown () {
  //driver.quit();
}
```

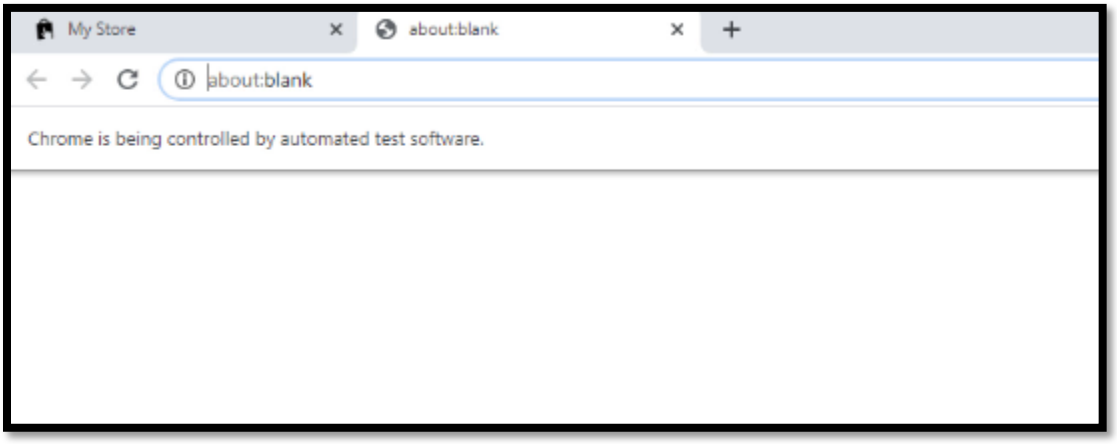
Let's start the test by writing @Test / public void testNewWindowConcept () { }
driver.switchTo().newWindow(WindowType.TAB);

```
@Test
public void testNewWindowConcept () {
  driver.switchTo().newWindow(WindowType.TAB);
}
```

If I hover the newWindow() method, and the description states "Creates a new browser window and switches the focus for future commands of this driver to the new window". In shorter words, it will open a new window then switch focus to that window. We also see the parameters are type hint which is a type of new browser window to be created. The type hint will be *WINDOW* or *TAB*. Returns this driver focused on the given window. When it says window, it's referring to a window or a tab. That's why I was able to write WindowType.TAB. One more point, do you see how the Return Type is WebDriver?



If we choose to, we can assign this statement to WebDriver with any name like newPage =. Let's Run. We have 2 tabs and a blank page is opened with automatic focus in the new tab.



The same outcome happens with a new window. Change TAB to WINDOW then run. A blank page is opened in a new window. For our Test Script, after the blank page, we are going to open the Contact Us page. Let me grab this URL and load a new page by writing newPage.get("Paste the URL") and print the page title sysout ("Title: " + driver.getTitle()).

```
@Test
Run | Debug
public void testNewWindowConcept () {
  WebDriver newPage = driver.switchTo().newWindow(WindowType.WINDOW);
  newPage.get("http://automationpractice.com/index.php?controller=contact");
  System.out.println("Title: " + driver.getTitle());
}
```

Let's Run. Both titles are displayed in the Console: My Store and Contact Us – My Store.

```
INFO: Detected dialect: W3C
Title: My Store
Title: Contact us - My Store
PASSED: testNewWindowConcept
```

Next, I will show you how to open, switch, work in the new tab then switch back to the parent tab.

Open, Switch & Work In New Tab Then Switch Back To Parent Tab

Our steps for opening, switching, working, and switching back to the parent is to access this Home page of this Automation Practice site then open the Sign In page in the new tab. On the Sign In page, we will work in this tab by entering an email address then clicking the Create An Account button. We are not going to perform any actions to create an account but switch back to the parent tab and start shopping.

Let's start our test by writing @Test / public void testOpenSwitchWorkInNewTab () {} then some comments as our steps. // Open & Switch To New Window-Tab
// Work In The New Window-Tab // Get The Window ID Handles // Switch Back To The Parent Window-Tab & Work In The 1st Window-Tab.

```
@Test
Run | Debug
public void testOpenSwitchWorkInNewTab () {

  // Open & Switch To New Window-Tab

  // Work In The New Window-Tab

  // Get The Window ID Handles

  // Switch Back To The Parent Window-Tab & Work In The 1st Window-Tab
}
```

First Step: We are going to Open & Switch To The New Window-Tab, we write driver.switchTo().newWindow(WindowType.TAB).get("http://automationpractice.com/index.php?controller=authentication&back=my-account"); Paste the URL. Since this URL is so long, we can place the get method on the next line my hitting enter after the dot. You can also end the previous line with a semi-colon and start the next line with driver.get. However, I'm going to remove the semi-colon and driver. It's still long. But let's print the title sysout("Title: " + driver.getTitle());

```
// Open & Switch To New Window-Tab
driver.switchTo().newWindow(WindowType.TAB).
get("http://automationpractice.com/index.php?controller=authentication&back=my-account");
System.out.println("Title: " + driver.getTitle());
```

Now, let's work in the new tab. Go back to the AUT and inspect both elements. Email address has email_create as the id value and the button has SubmitCreate as the id value. Capital S and Capital C. Okay, for the email we write

driver.findElement(By.id(“email_create”)).sendKeys(“Automation34@Selenium4.com”); then driver.findElement(By.id(“SubmitCreate”)).click(); for the button.

```
// Work In The New Window-Tab
driver.findElement(By.id("email_create")).sendKeys("Automation34@Selenium4.com");
driver.findElement(By.id("SubmitCreate")).click();
```

We are finished working in the new tab. At this point, you can decide to close the new tab or leave it open. If you close the tab, make sure to use driver.close and not driver.quit. driver.close will close the new window but driver.quit will quit the driver and close both windows. Video 18 that will show you the difference between close and quit.

Now, let’s get the Window Handles of each tab. The Window Handle is an alphanumeric id assigned to each window or each tab. Set <String> allWindows = driver.getWindow and we 2 methods. The getWindowHandle method returns 1 window id handle. getWindowHandles method return a set of window handles that can be used to iterate over all open windows. Select this method. The next method is to return an Iterator for the collection of both Window ID’s. <String> iterate = allWindows.iterator(); We need a way to iterate the collection of elements and next() is a method for returning an element. String parentFirstWindow = is the parent tab and first element in the iteration. iterate dot is the next() element which is String childSecondWindow = and that window is the new tab we are going to open. For this example, that’s all we need to get the ID’s and iterate the ID’s. You can also loop through the windows.

```
// Get The Window ID Handles
Set <String> allWindows = driver.getWindowHandles();
Iterator<String> iterate = allWindows.iterator();
String parentFirstWindow = iterate.next();
String childSecondWindow = iterate.next();
```

The last step is to switch back to the parent. Let’s write driver.switchTo().window. Let me explain this part. Do you see window and newWindow? Both methods involve switching focus to a window but the window() method deals with handling a window and the newWindow() method opens a window. The window() method receives a name or handle. We get the name or handle information after using the getWindowHandle method or getWindowHandles method. There’s a difference between handling the window and opening the window. I created video 71 that provide more details on how to handle the windows. In that video, I used all 3 methods: getWindowHandle, getWindowHandles, and driver.switchTo().window().

Right now, we are opening a new window then working in the window which is a benefit because it allows us to multi-task in a different tab. Before Selenium 4, we could not multi-task but had to perform a different task in the same tab. Select window then pass in parentFirstWindow because we are switching back to the parent. Let's go to the AUT and inspect the Search box. We see id is search_query_top and the search icon has no id value but a value of submit_search for name.

Let's enter some data by writing driver.findElement(By.id("search_query_top")).sendKeys("Faded Shirt"); and click the search icon by writing driver.findElement(By.name("submit_search")).click; Also print the page title.
sysout("Title: " + driver.getTitle());

```
// Switch Back To The Parent Window-Tab & Work In The 1st Window-Tab
driver.switchTo().window(parentFirstWindow);
driver.findElement(By.id("search_query_top")).sendKeys("Faded Shirt");
driver.findElement(By.name("submit_search")).click();
System.out.println("Title: " + driver.getTitle());
```

Let's Run. The reason why I like to print the title because I want to make sure we can see the page title for each page in the console. We see both tabs had some kind of information. The parent tab has Faded Shirt and the new tab is ready to create an account with our Email Address already populated.

Let's see if the console shows each page title. The console shows all 3 titles: My Store, Login – My Store, and Search – My Store. Bingo – Thanks for watching.

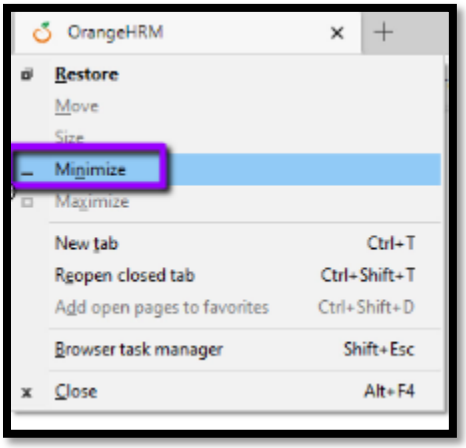
Minimize Window

In this session, let's look at a new Selenium 4 feature to minimize the browser. We could always minimize the browser but this feature is built into Selenium. The other ways were to set the position of a window and to use the Robot Class. Personally, I like to watch the automation of our Test Scripts in a browser but if you have a requirement for minimizing the browser then this will help you out.

Also, if you are interested in more videos, you can subscribe to my YouTube channel and click the bell icon. Also, follow me on Twitter, connect with me on LinkedIn and Facebook.

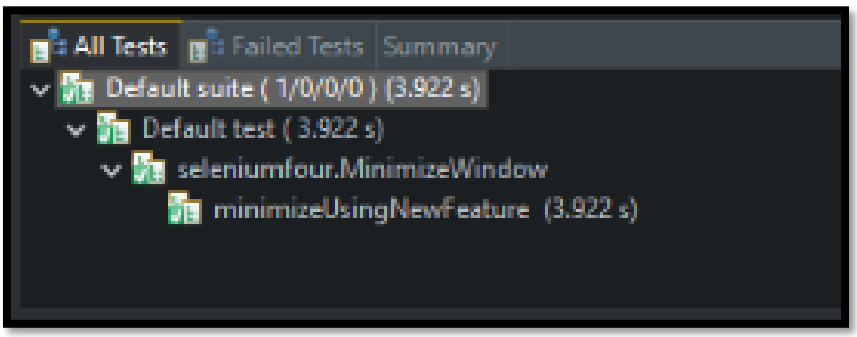
We are going to use this Orange HRM site by entering the username, password, and clicking the button.

Username is Admin and Password is admin123. Inspect the Username and it shows txtUsername as the value for id, inspect the Password and it shows txtPassword as the value for id. Now, let's inspect the button and it shows btnLogin as the value for id. To use the Robot Class, we select ALT + SPACE + N and do you see Minimize? That's how we minimize using the Robot class.



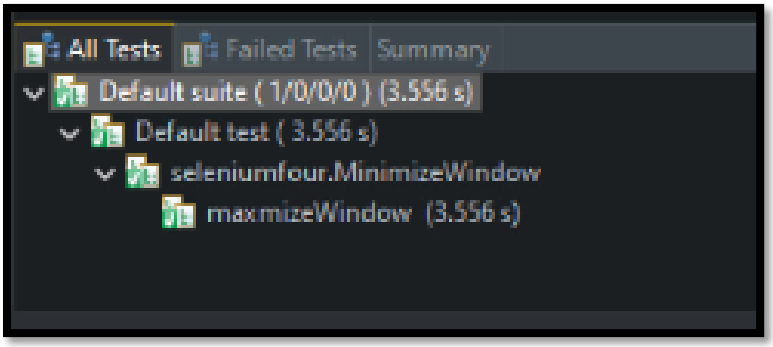
Oh, by the way, this minimize feature became available in Selenium 4 Alpha 5. So far, this Test Script is set up for ChromeDriver and to load our AUT. Now, we write @Test
public void minimizeUsingNewFeature () {}
Import the @Test annotation from TestNG. Now, we write driver.manage().window(). Normally we select maximize but here's minimize(). Enter the username driver.findElement(By.id("txtUsername")).sendKeys("Admin");
Enter the password driver.findElement(By.id("txtPassword")).sendKeys("admin123");
Next is to click the button driver.findElement(By.id("btnLogin")).click(); Let's Run. It passed in 3.922 seconds.

```
@Test
Run | Debug
public void minimizeUsingNewFeature () {
  driver.manage().window().minimize();
  driver.findElement(By.id("txtUsername")).sendKeys("Admin");
  driver.findElement(By.id("txtPassword")).sendKeys("admin123");
  driver.findElement(By.id("btnLogin")).click();
}
```



I've seen it runs faster with minimize and also seen it run faster with maximize. Let's run it with maximize and see what happens. @Test
public void maximizeWindow () {} Copy and Paste this code from minimize and change to minimize to maximize. Run. Maximize executed in 3.556 seconds.

```
@Test
Run | Debug
public void maxmizeWindow () {
  driver.manage().window().maximize();
  driver.findElement(By.id("txtUsername")).sendKeys("Admin");
  driver.findElement(By.id("txtPassword")).sendKeys("admin123");
  driver.findElement(By.id("btnLogin")).click();
}
```



Now, let's minimize the browser by setting the position and using the Robot class

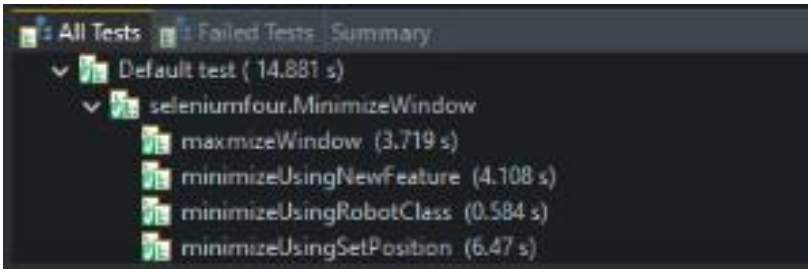
Starting with set position, we write @Test public void minimizeUsingSetPosition () {}
driver.manage().window().setPosition(new Point(-2000, 0)); Import Point from the Selenium package. Point is a Selenium class that's a copy of Java's awt.Point package. It removed the dependency of awt. Copy and paste the same code. We're finished with the Set Position test.

```
@Test
Run | Debug
public void minimizeUsingSetPosition () {
  driver.manage().window().setPosition(new Point(-2000, 0));
  driver.findElement(By.id("txtUsername")).sendKeys("Admin");
  driver.findElement(By.id("txtPassword")).sendKeys("admin123");
  driver.findElement(By.id("btnLogin")).click();
}
```

Next is the Robot class. @Test public void minimizeUsingRobotClass () {} Robot robot = new Robot (); Import Robot from java.awt then add throws declaration – select AWTException but we can also select Exception. First, we press the ALT Key by entering robot.keyPress(KeyEvent.VK_ALT); Now, let's press the SPACE key. robot.keyPress(KeyEvent.VK_SPACE); Next is the N key robot.keyPress(KeyEvent.VK_N); We have to also release the keys so I'm going to copy these 3 statements, paste them and change Press to Release. For the last time, change Press to Release.

```
@Test
Run | Debug
public void minimizeUsingRobotClass () throws AWTException {
  Robot robot = new Robot ();
  robot.keyPress(KeyEvent.VK_ALT);
  robot.keyPress(KeyEvent.VK_SPACE);
  robot.keyPress(KeyEvent.VK_N);

  robot.keyRelease(KeyEvent.VK_ALT);
  robot.keyRelease(KeyEvent.VK_SPACE);
  robot.keyRelease(KeyEvent.VK_N);
```

Now, let's run all 4 of the methods: minimizUsingRobotClass, minimizeUsingSetPosition, maximizeWindow, and minimizeUsingNewFeature. Run All. We see the time for each Test Script and it looks like Robot class was the fastest. This time it showed maximize 2nd, minimize 3rd, and minimize using set position as the last for as speed Thanks for watching and I'll see you in the next session.

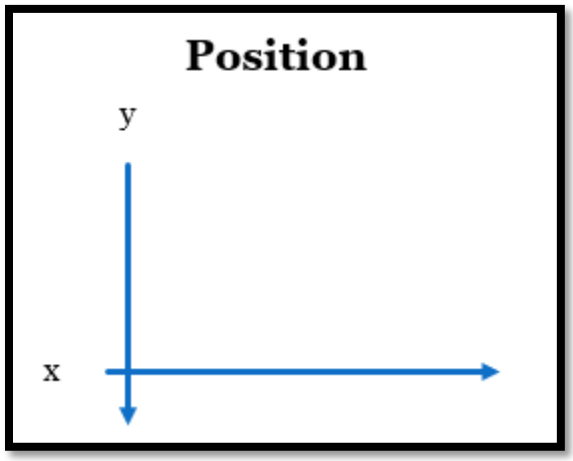

All Tests
Failed Tests
Summary
Default test (14.881 s)
seleniumfour.MinimizeWindow
maximizeWindow (3.719 s)
minimizeUsingNewFeature (4.108 s)
minimizeUsingRobotClass (0.584 s)
minimizeUsingSetPositon (6.47 s)

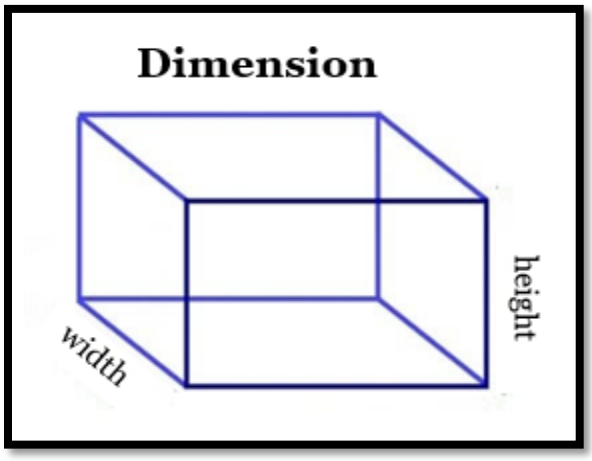
getRect

Introduction

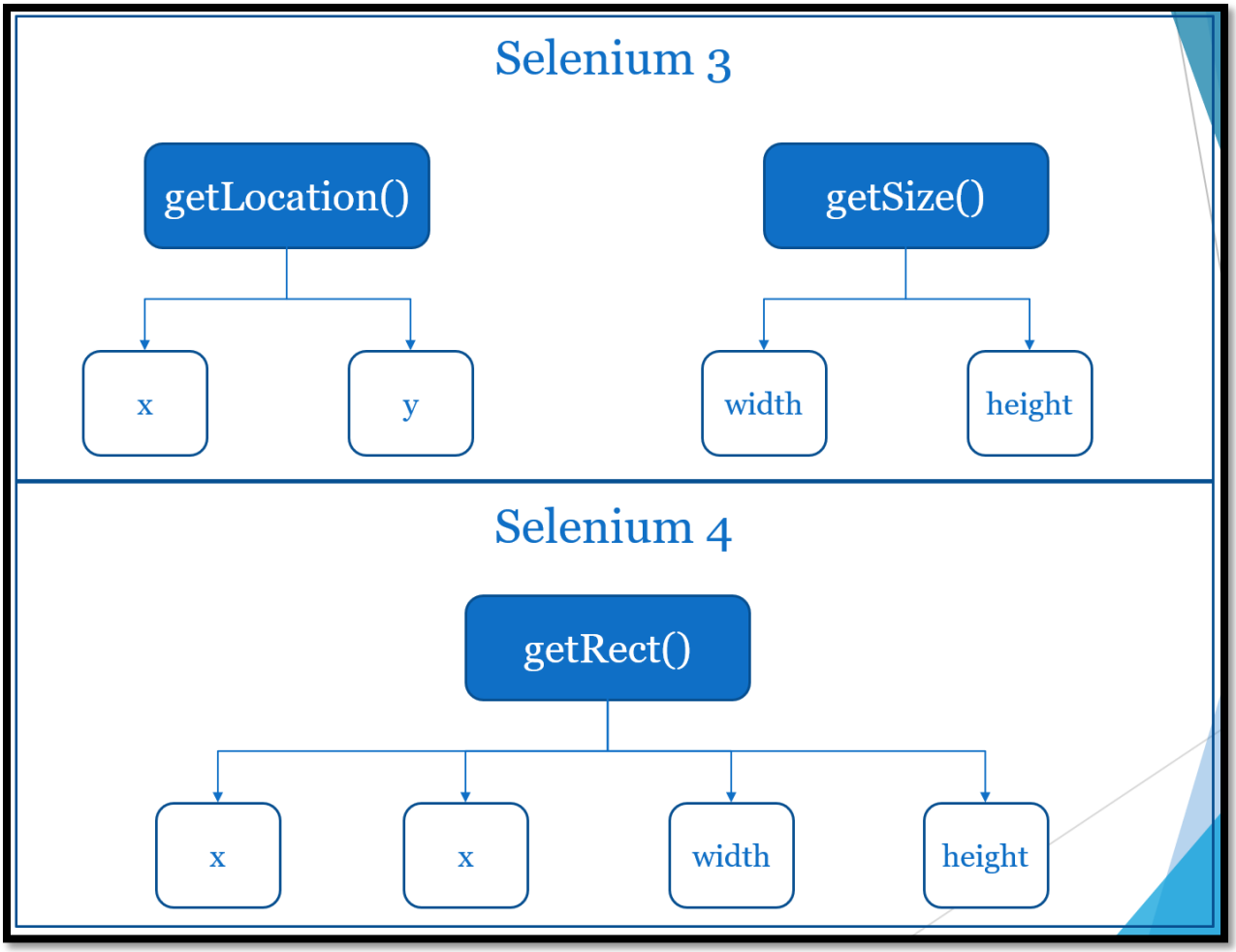
In our automation program, there are scenarios that requires us to get the position and/or dimension of a WebElement. The positions are x and y coordinates while the dimensions are width and height. Selenium 4 has the getRect method that allows to get both positions and dimensions.

x comes before y and width comes before height. The x coordinate is measured horizontally starting from the left to the right. The y coordinate is measured vertically starting from the top to the bottom. Width and height are ways to measure the size of an element. Notice, there are 2 differences between Selenium and how we normally measure in Math. On a web page, the length is not available as a Dimension and y is not measured starting from the bottom.



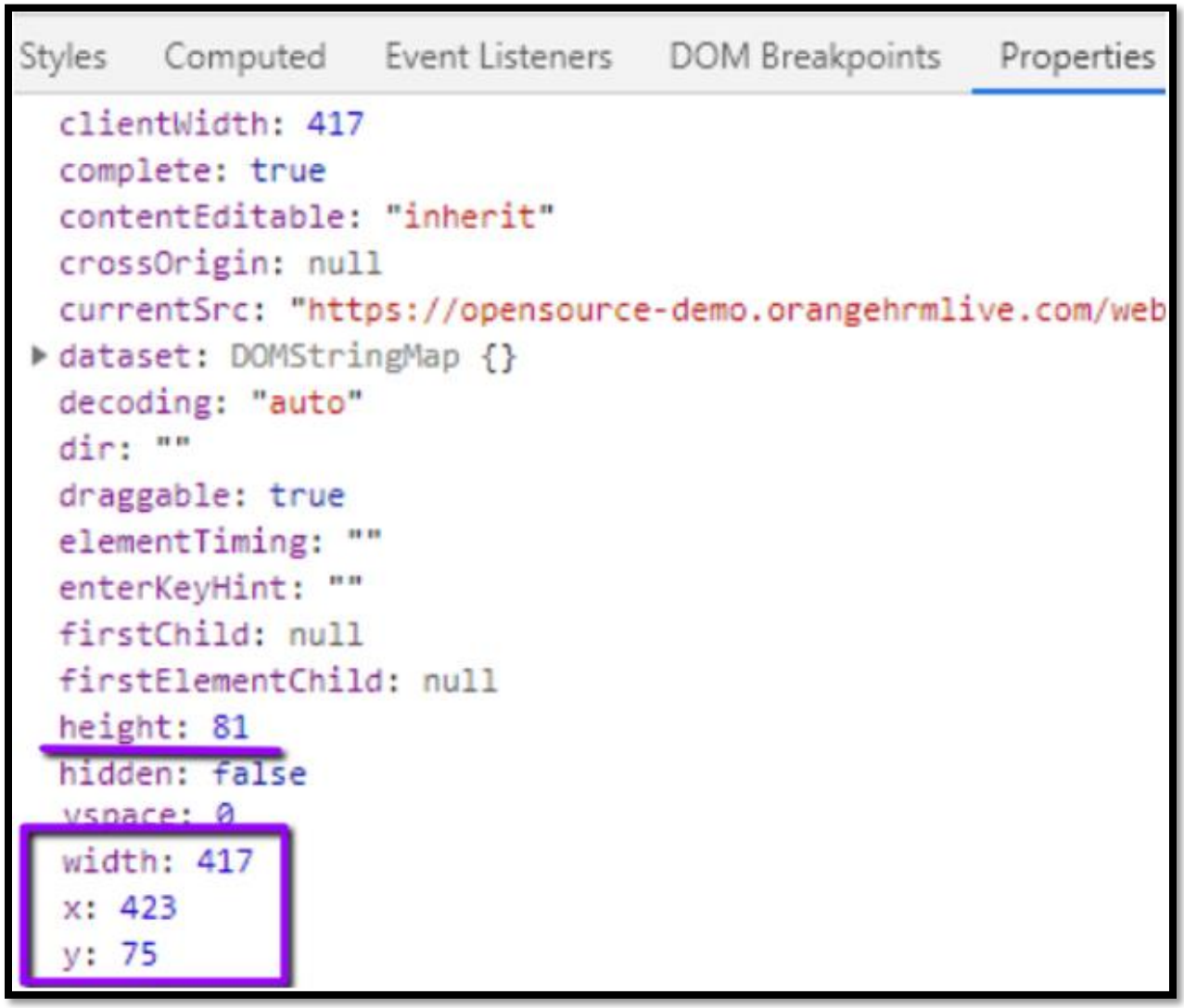


With Selenium 3, we have getLocation and getSize. The getLocation method returns x and it returns y while the getSize method is used to fetch the width and height. With Selenium 4, the new getRect method is a combination that gets the x coordinate, y coordinate, width, and height. In this video, I am going to show you all 3 methods: getRect, getLocation, and getSize then compare getRect to see if it returns the same value as getLocation and getSize.



If you are interested in programming and automation videos, subscribe to my YouTube channel then click the bell icon. Also, connect with me on LinkedIn, Facebook, and follow me on Twitter. The code, presentation, and transcript document will get placed on GitHub.

We are going to use this Orange HRM logo as our WebElement. When I inspect the logo, we see img as the tag name and src as the attribute but no id attribute. However, the parent has an id value of divLogo. There are different ways to get this value. This time, I'm going to use CSS. CTRL + F to bring up this Find by string, selector, or XPath then write hash tag # then divLogo > img. Bingo. Copy this value. Also, let's look at the size and location. Go to Properties, select img and we see height as 81, width is 417, x is 423, and y is 75.



getRect

Let's go to our Test Script and start with the getRect method to make sure the measurements are the same. @Test / public void getPositionDimension () {}

The WebElement is logoOrangeHRM = driver.findElement(By.cssSelector("#divLogo > img")); Paste the value. Now that we have the logoOrangeHRM. WebElement, we can access the getRect method. Do you see Rectangle? That means the getRect method returns a Rectangle. The description shows it returns the location and size of the rendered element. Since it returns a Rectangle, we assign the location and size to Rectangle with an object name like rectLogo =. That's all we need to get the x and y coordinates plus the width and height. Import the Rectangle class from Selenium package. Also import the @Test annotation from TestNG.

Let's print these values and start with x. sysout("x: " + rectLogo.get) and we see getHeight, getWidth, getX, and getY. Select getX then perform this same process for each method. sysout("y: " + rectLogo.getY()); sysout("Width: " + rectLogo.getWidth()); sysout("Height: " + rectLogo.getHeight());

```
@Test
Run | Debug
public void getPositionDimension () {
  WebElement logoOrangeHRM = driver.findElement(By.cssSelector("#divLogo > img"));
  Rectangle rectLogo = logoOrangeHRM.getRect();
  System.out.println("x: " + rectLogo.getX());
  System.out.println("y: " + rectLogo.getY());
  System.out.println("Width: " + rectLogo.getWidth());
  System.out.println("Height: " + rectLogo.getHeight());
}
```

Let's Run. It's the same values. x = 423, y = 75, Width = 417, Height = 81

```
INFO: Detected dialect: W3C
x: 423
y: 75
Width: 417
Height: 81
PASSED: getPositionDimension
```

Let me hover over this Rectangle class then open the Declaration, we see 2 constructors. The 1st constructor has 4 int parameters: x, y, height, and width. However, the 2nd constructor has a Point parameter and a Dimension. With Selenium 3, we had to use the Point and Dimension classes to help us get the location and size.

```
public class Rectangle {

  public int x;
  public int y;
  public int height;
  public int width;

  public Rectangle(int x, int y, int height, int width) {
    this.x = x;
    this.y = y;
    this.height = height;
    this.width = width;
  }

  public Rectangle(Point p, Dimension d) {
    x = p.x;
    y = p.y;
    height = d.height;
    width = d.width;
  }
```

Both classes are still available in Selenium 3. Let's compare getRect with getLocation and getSize.

getLocation & getSize

Go back to our Test Script and Erase these print statements then start from scratch. For location, we write logoOrangeHRM.getLocation. and we see Point as the Return Type. After selecting getLocation, we assign it to Point with any name like pointLogo =. Import the class from Selenium. Look at this, we write pointLogo., we see x, y, getX, and getY but notice, we do not have width and height. Width and height are not available because this is the Point class.

For width and height, we write the WebElement logoOrangeHRM.getSize. This time, we see Dimension as the return type. So, we select getSize then assign it to Dimension with dimLogo = as the object. Import Dimension from the Selenium class. Now, dimLogo. have height, width, getHeight, and getWidth.

Let's print the values to verify we get the same values. Let's add some headers. sysout("\t Selenium 3 \t Selenium 4"); Now, let' add some hyphens. sysout("----------") All of the methods came from Selenium 4 but I put Selenium 3 in the print statement. Selenium 3 will represent the old way of getting the location and getting the size. Let's add some print statements to get x, y, width, and height. Start with x.
sysout("x \t\t " + pointLogo.getX() + "\t\t" + rectLogo.getX());
sysout("y \t\t " + pointLogo.getY() + "\t\t" + rectLogo.getY());
sysout("Width \t\t " + dimLogo.getWidth() + "\t\t" + rectLogo.getWidth);
sysout("Height \t\t " + dimLogo.getHeight() + "\t\t" + rectLogo.getHeight);

```
@Test
Run | Debug
public void getPositionDimension () {
  WebElement logoOrangeHRM = driver.findElement(By.cssSelector("#divLogo > img"));
  Rectangle rectLogo = logoOrangeHRM.getRect();
  Point pointLogo = logoOrangeHRM.getLocation();
  Dimension dimLogo = logoOrangeHRM.getSize();
  System.out.println("\t    Selenium 3 \t   Selenium 4");
  System.out.println("-----------------------------------------------");
  System.out.println("x \t\t" + pointLogo.getX() + "\t\t" + rectLogo.getX());
  System.out.println("y \t\t" + pointLogo.getY() + "\t\t" + rectLogo.getY());
  System.out.println("Width \t\t" + dimLogo.getWidth() + "\t\t" + rectLogo.getWidth());
  System.out.println("Height \t\t" + dimLogo.getHeight() + "\t\t" + rectLogo.getHeight());
}
```

Let's Run. We see the same exact value for x, y, Width, and Height.

```
INFO: Detected dialect: W3C
            Selenium 3     Selenium 4
-----------------------------------------------
x               423             423
y               75              75
Width           417             417
Height          81              81
PASSED: getPositionDimension
```

The benefit of Selenium 4's Rectangle class is that it has all of the methods. That's it and Thanks for watching and I'll see you in the next session.

WebElement Screenshot

In this session, we are going to take a screenshot of a WebElement. Until now, Selenium already had a feature to take a screenshot of a page but not to take a screenshot of a WebElement. Before Selenium 4, we had to use an API called Ashot to take a screenshot of a WebElement. However, in this session I am going to demo how to take a screenshot of 1 WebElement and page section that includes more than 1 WebElement.

If you are interested in more videos, you can subscribe to my YouTube channel then click the bell icon. Follow me on Twitter and connect with me on LinkedIn and Facebook. Also, I'm going to place the transcript and code on GitHub.

We are going to use this Orange HRM site. I will take a screenshot of this logo for 1 WebElement. Do you see how this part of the page has more than 1 WebElement? We see Username, Password, Login Button, and the Forgot Password link plus this Orange image? I'm going to take screenshot of this entire section using Selenium 4. Let's start with the logo and Inspect then find the value by writing #divLogo. This is the id value for the parent > img is the tag name for the logo. Let's start by taking a screenshot of the WebElement logo.

One WebElement Screenshot

We have our test setup to load Chrome and the AUT.

```
@BeforeClass
public void setUp () {
 WebDriverManager.chromedriver().setup();
 driver = new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://opensource-demo.orangehrmlive.com/");
}

@AfterClass
public void tearDown () {
 driver.quit();
}
```

Now, let's take a screenshot of the logo by writing @Test / public void takeWebElementScreenshot () { } Let's find the WebElement by writing WebElement logoOrangeHRM = driver.findElement(By.cssSelector("#divLogo > img")); Import the WebElement and @Test annotation

from TestNG. Now that we have the logo for OrangeHRM logoOrangeHRM., we get the screenshotAs(OutputType.FILE). The getScreenshotAs method Capture the screenshot and store it in the specified location. Notice the return type is File. Let's hover the OutputType and it defines the output type for a screenshot. FILE is used to obtain the screenshot into a temporary file. Therefore, we assign the screenshot to a File and name it source =.

Now we need a File for our destination = new File called ("Orange HRM Logo.png"); We have our source file and destination file. A class called FileUtils. has a method to copyFile. The description shows it copies a file to a new location. So, for the source file we are going to have source and the destination file we are going to have our destination. Depending on how your Selenium 4 is setup. You may have to download the Apache Commons IO jar from Maven's Repository to get this FileUtils class then import the jar. It's also a dependency hierarchy for WebDriverManager. Last step is to add the throws declaration for IOException.

```
@Test
Run | Debug
public void takeWebElementScreenshot () throws IOException {
  WebElement logoOrangeHRM = driver.findElement(By.cssSelector("#divLogo > img"));
  File source = logoOrangeHRM.getScreenshotAs(OutputType.FILE);
  File destination = new File ("Orange HRM Logo.png");
  FileUtils.copyFile(source, destination);
}
```

That's it. Let's Run. We see the screenshot in our project. I setup Eclipse to auto refresh so I did not have to manually right click the project and select F5. Video 72 will show you how to Auto Refresh your project automatically. Right click, select Open, and here's the logo WebElement screenshot.

Next is to take a screenshot of a page section.

Page Section Screenshot

In a section, there are multiple WebElements so we need to find the parent tag of those WebElements. Inspect this section and we see the parent tag has an id value of divLoginImage. On this page, another section is the Social Media icons that can also be used to take a screenshot of multiple WebElements. All we need to find is the parent tag and Selenium will take a screenshot of all 4 Social Media icons.

In this example, social-icons is the parent tag. Go back to our IDE and write our test for the page section. So, I'm going to write @Test / public void takePageSectionScreenshot () { } The page section is a WebElement so we write WebElement pageSectionOrangeHRM = driver.findElement(By.id("divLoginImage")); Next is to store the screenshot in a file. Therefore, we write File source = What is the source? The source pageSectionOrangeHRM.getScreenshotAs(OutputType.FILE);

This statement is an abstract representation of the File class and will save the screenshot in memory. We need a physical file since this is an abstract representation. Therefore, we copy the source file to the destination. I'm going to use these 2 statements and combine them into 1 line. So, I'm going to write FileUtils.copyFile(source,). The destination file will contain the name of the new File("Orange HRM Page Section.png"). You can also use .jpg file but I like to use png. This statement will convert the abstract source file into a physical .png file so we can view the screenshot. Add the throws declaration and Run. We see the Page Section screenshot. Let's open and all of the WebElements are located in this screenshot.

That's it and Thanks for watching.

Firefox Full Page Screenshot

Introduction

In this session, I am going to take a full-page screenshot using Selenium 4. Right now, it's only available for Firefox. There's a difference between the full-page screenshot and the existing screenshot feature. The existing screenshot feature takes a screenshot of the page we can see in our web page. However, the full-page screenshot takes a screenshot of the entire page whether we can see it or not.

I create videos every week and if you are interested in more videos, feel free to connect with me on LinkedIn and Facebook. You can also follow me on Twitter and GitHub. If you're on YouTube, you can subscribe to my YouTube channel and click the bell icon. I'll make sure to place the transcript and code on GitHub.

Our AUT is going to be Amazon. On this page, we see it's very long and has a lot of products. Scroll and scroll and we see more and more. Now, let's go to our code.

Full Page Screenshot

We have a setup method for Amazon using FirefoxDriver and a tear down method to quit the driver.

```
@BeforeTest
public void setUp () {
 WebDriverManager.firefoxdriver().setup();
 driver = new FirefoxDriver ();
 driver.manage().window().maximize();
 driver.get("https://www.amazon.com/");
}

@AfterTest
public void tearDown () {
 driver.quit();
}
```

Start by writing @Test / public void takeFullPageScreenshot () { }. The 1st step is to capture the screenshot using ((FirefoxDriver)driver). We are casting FirefoxDriver so this statement is a downcast. The plan is to use driver which is a variable of type WebDriver. We are going to use driver to call a method that's only available to FirefoxDriver and that method is getFullPageScreenshotAs(OutputType.FILE); Let's save the FILE and name it source =. Import the @Test Annotation from TestNG and File.

If you want, feel free to separate this same statement into 2 statements. However, I'm going continue getting straight to the point. Also, with the next statement we can separate into 2 separate statements but I will write 1 line. Let's handle the file with FileHandler.copy(source,). The destination is a new File() named ("Amazon Full Page Screenshot.png"); That's all we write to capture the full page. Add a throws declaration and select IOException.

```
@Test
Run | Debug
public void takeFullPageScreenshot () throws IOException {
  File source = ((FirefoxDriver)driver).getFullPageScreenshotAs(OutputType.FILE);
  FileHandler.copy(source, new File("Amazon Full Page Screenshot.png"));
}
```

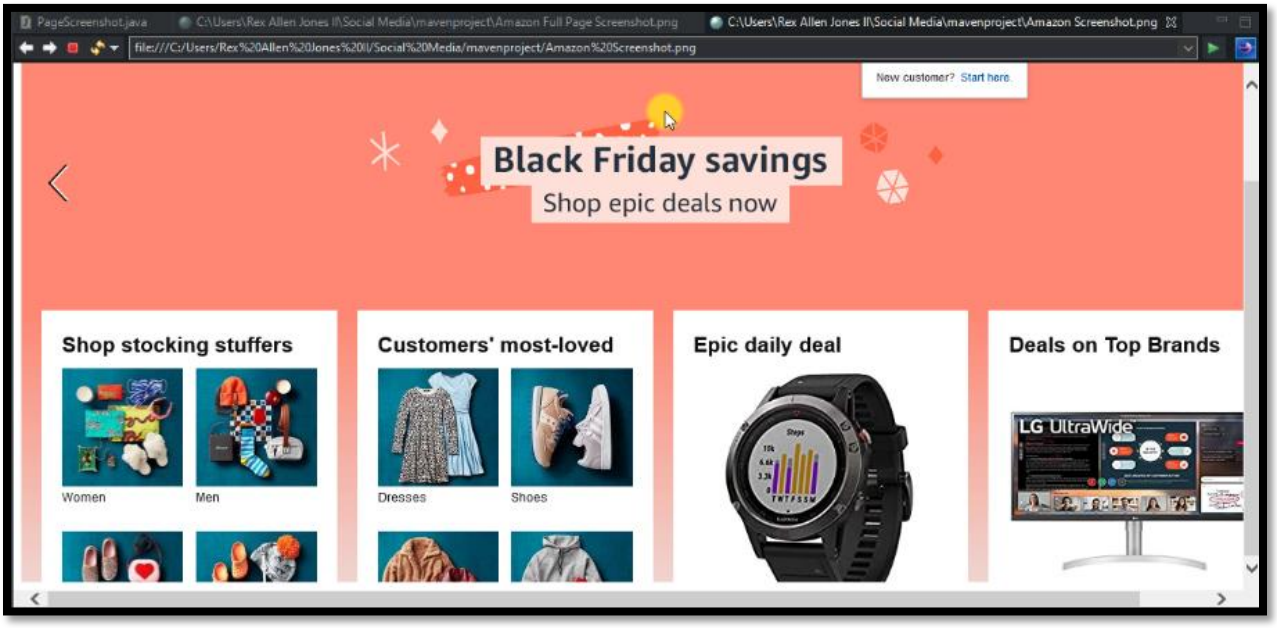
Let me also capture a screenshot but not the full page. It won't take long. I want to show you the difference between full page screenshot and the existing screenshot feature.

@Test / public void takePageScreenshot () { } It's very similar to the full page screenshot but I will write TakesScreenshot and not FirefixDriver. Alright, File source = then we cast ((TakesScreenhot)driver). Import TakesScreenshot from Selenium package and .screenshot notice we only see getScreenshotAs but we do not see getFullPageScreenshotAs. That's how we know getFullPageScreenshotAs is only available for the FirefoxDriver. Select getScreenshotAs(OutputType.FILE). Next, is File CTRL + SPACE. We have the option of using FileHandler or FileUtils. FileHandler is a class from Selenium and FileUtils is a class from apache.commons. I'm going to use FileUtils this time since I used FileHandler in the previous Test Script and select copyFile(source, new File("Amazon Screenshot.png")); Import throws declaration and we are going to select IOException.

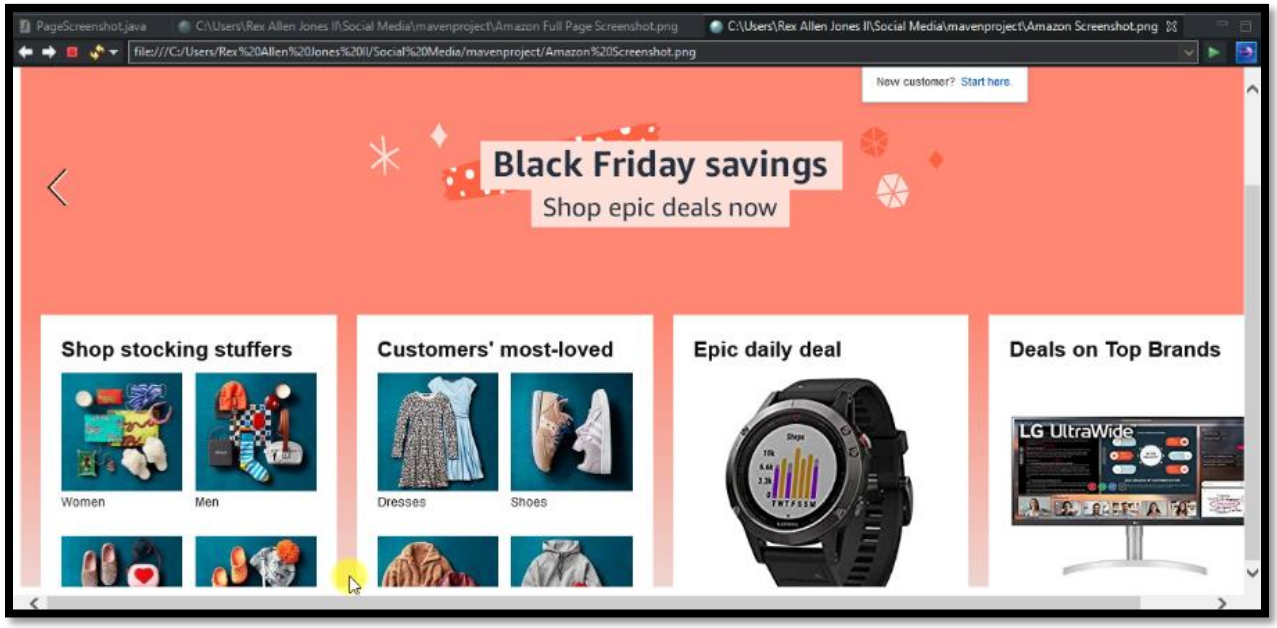
```
@Test
Run | Debug
public void takePageScreenshot () throws IOException {
  File source = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
  FileUtils.copyFile(source, new File("Amazon Screenshot.png"));
}
```

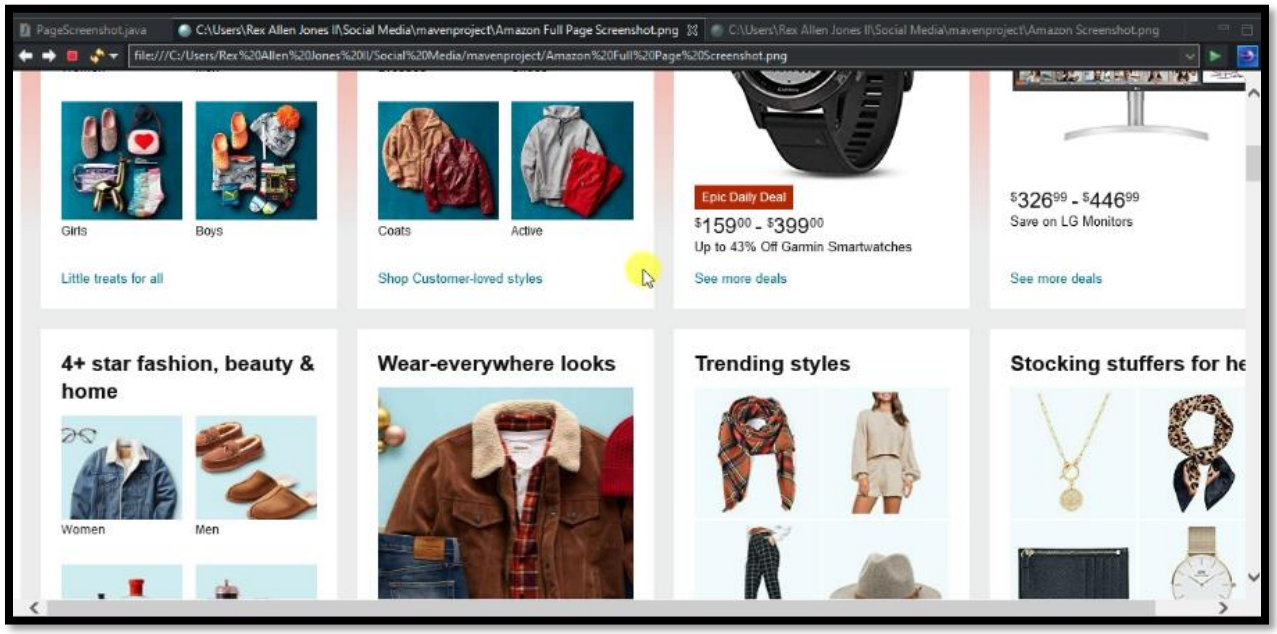
I'm going to save it and this time Run All and not an individual Test Script. We see both screenshots: Amazon Full Page Screenshot and Amazon Screenshot. Like I mentioned in the previous video, I set it up where it refreshes automatically.

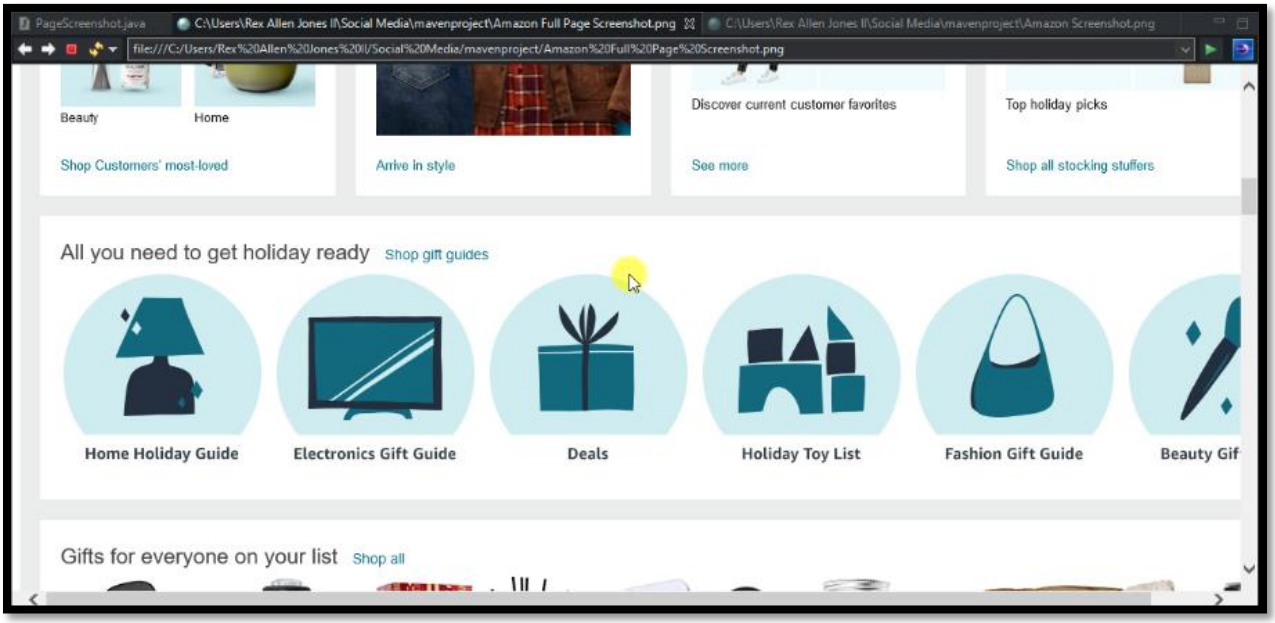
This here is the screenshot which shows what we only saw in the webpage and it stops around this point.

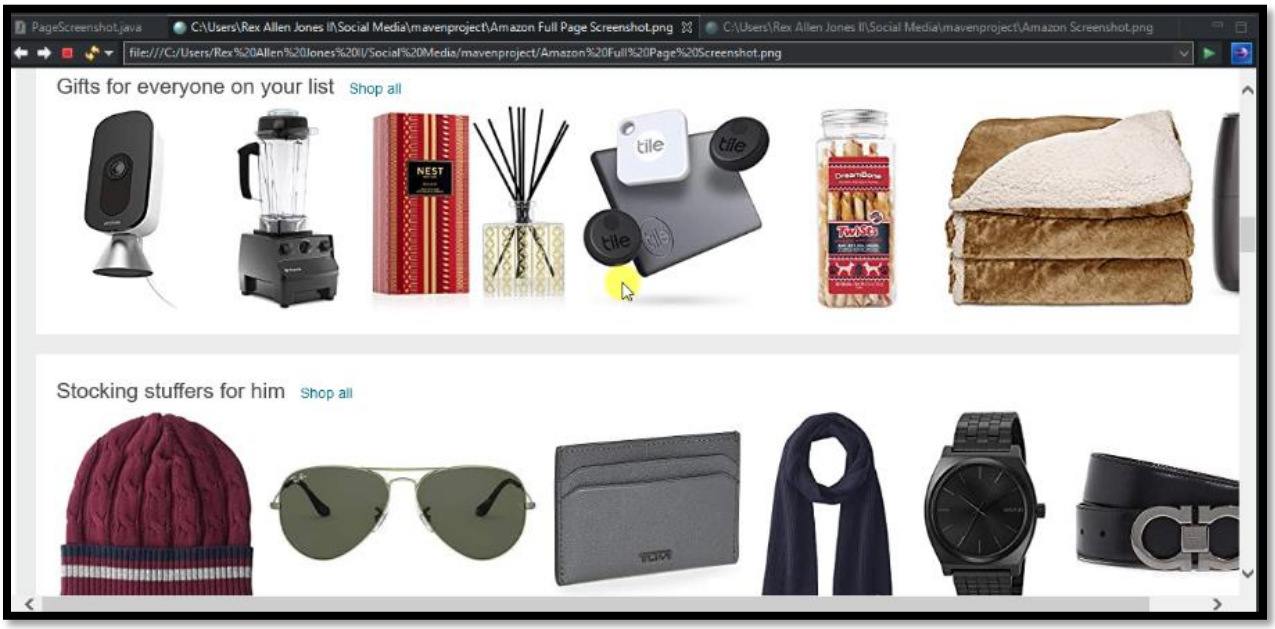


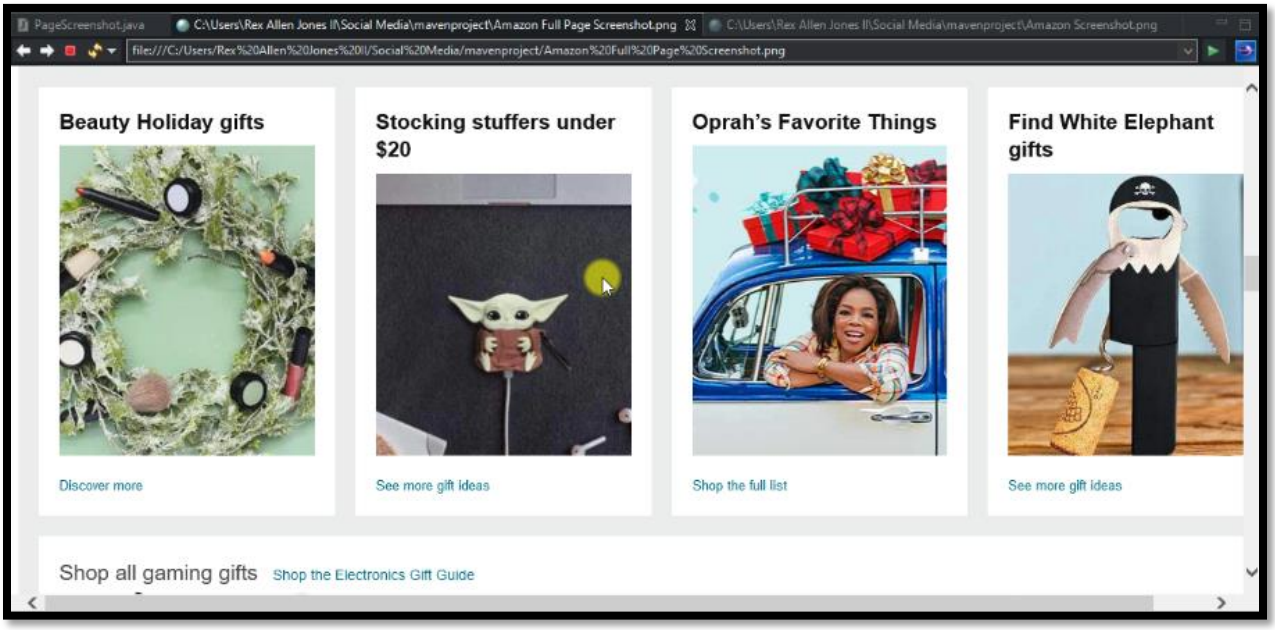
Select the other screenshot and we keep on scrolling and the full-page screenshot took a screenshot of the complete page from top to bottom but not including the tab and address bar. Taking a normal screenshot only captured what we see in the browser page. That's it and thank you for watching and I'll see you in the next session.

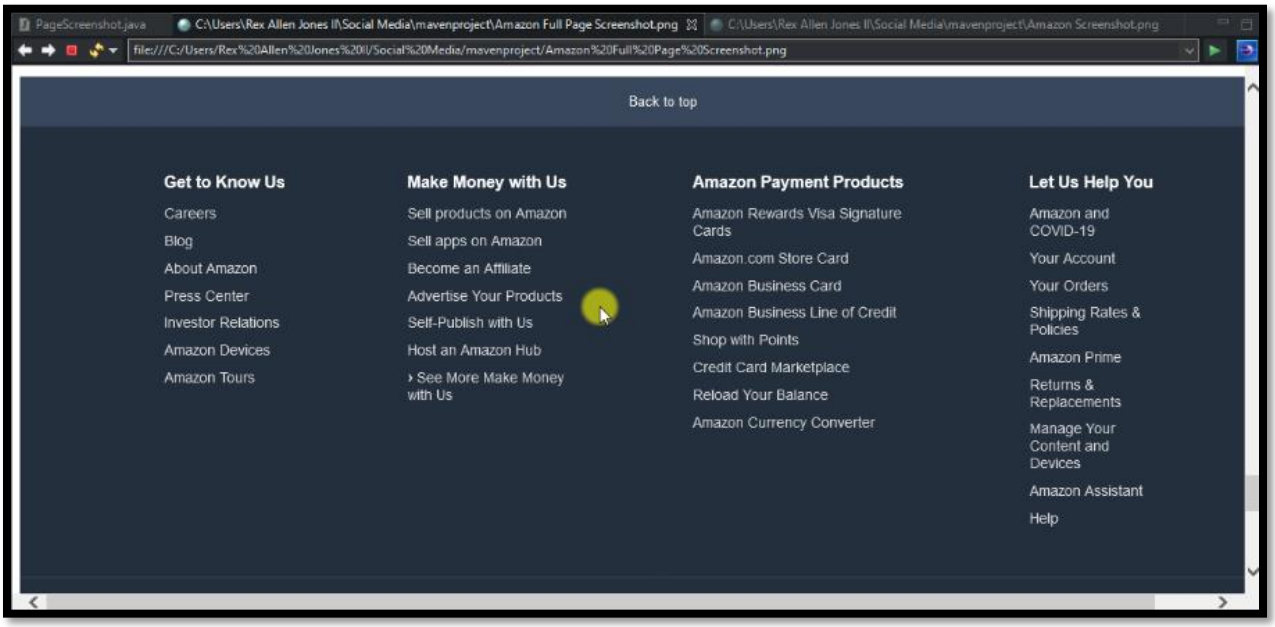

PageScreenshot.java
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Full Page Screenshot.png
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Screenshot.png
file:///C:/Users/Rex%20Allen%20Jones%20II/Social%20Media/mavenproject/Amazon%20Screenshot.png
New customer? Start here.
Black Friday savings
Shop epic deals now
Shop stocking stuffers
Women
Men
Customers' most-loved
Dresses
Shoes
Epic daily deal
Deals on Top Brands
LG UltraWide


PageScreenshot.java
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Full Page Screenshot.png
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Screenshot.png
file:///C:/Users/Rex%20Allen%20Jones%20II/Social%20Media/mavenproject/Amazon%20Full%20Page%20Screenshot.png
Girls
Boys
Little treats for all
Coats
Active
Shop Customer-loved styles
Epic Daily Deal
$159.00 - $399.00
Up to 43% Off Garmin Smartwatches
See more deals
$326.99 - $446.99
Save on LG Monitors
See more deals
4+ star fashion, beauty & home
Women
Men
Wear-everywhere looks
Trending styles
Stocking stuffers for he


PageScreenshot.java
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Full Page Screenshot.png
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Screenshot.png
file:///C:/Users/Rex%20Allen%20Jones%20II/Social%20Media/mavenproject/Amazon%20Full%20Page%20Screenshot.png
Beauty
Home
Shop Customers' most-loved
Arrive in style
Discover current customer favorites
See more
Top holiday picks
Shop all stocking stuffers
All you need to get holiday ready
Shop gift guides
Home Holiday Guide
Electronics Gift Guide
Deals
Holiday Toy List
Fashion Gift Guide
Beauty Gif
Gifts for everyone on your list
Shop all


PageScreenshot.java
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Full Page Screenshot.png
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Screenshot.png
file:///C:/Users/Rex%20Allen%20Jones%20II/Social%20Media/mavenproject/Amazon%20Full%20Page%20Screenshot.png
Gifts for everyone on your list
Shop all
NEST
tile
Stocking stuffers for him
Shop all


PageScreenshot.java
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Full Page Screenshot.png
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Screenshot.png
file:///C:/Users/Rex%20Allen%20Jones%20II/Social%20Media/mavenproject/Amazon%20Full%20Page%20Screenshot.png
Beauty Holiday gifts
Discover more
Stocking stuffers under $20
See more gift ideas
Oprah's Favorite Things
Shop the full list
Find White Elephant gifts
See more gift ideas
Shop all gaming gifts
Shop the Electronics Gift Guide


PageScreenshot.java
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Full Page Screenshot.png
C:\Users\Rex Allen Jones II\Social Media\mavenproject\Amazon Screenshot.png
file:///C:/Users/Rex%20Allen%20Jones%20II/Social%20Media/mavenproject/Amazon%20Full%20Page%20Screenshot.png
Back to top
Get to Know Us
Careers
Blog
About Amazon
Press Center
Investor Relations
Amazon Devices
Amazon Tours
Make Money with Us
Sell products on Amazon
Sell apps on Amazon
Become an Affiliate
Advertise Your Products
Self-Publish with Us
Host an Amazon Hub
› See More Make Money with Us
Amazon Payment Products
Amazon Rewards Visa Signature Cards
Amazon.com Store Card
Amazon Business Card
Amazon Business Line of Credit
Shop with Points
Credit Card Marketplace
Reload Your Balance
Amazon Currency Converter
Let Us Help You
Amazon and COVID-19
Your Account
Your Orders
Shipping Rates & Policies
Amazon Prime
Returns & Replacements
Manage Your Content and Devices
Amazon Assistant
Help

Chrome Debugging Protocol

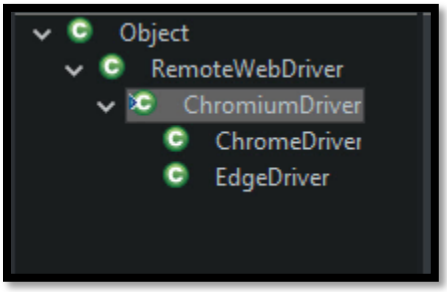
Introduction

Hello and Welcome, in this session, I will talk about CDP which is an acronym for Chrome Debugging Protocol. It's a new Selenium 4 API feature that's designed for debuggers. All browsers built on the Chromium platform has an option for Developer Tools. Therefore, another name for CDP can be called Chrome DevTools Protocol.

If I go to Chrome then click this vertical ellipsis, navigate to More Tools, and at this point, we see Developer Tools. DevTools is short for Developer Tools. Click and we see a few tabs. Elements is the most popular tab for Automation but we also have Console, Sources, Network, Performance, Memory Application, and Security. It's the same panel if I right click a page and click Inspect.

Microsoft Edge also has these same tabs after accessing Developer Tools. We click this ellipsis, More Tools and here's Developer Tools.

Selenium 4 provides a way for us to take advantage of Chrome and Microsoft Edge's debugging protocol. Let's look behind the scenes at the classes and methods. Starting with selenium-chromium-driver, we see the ChromiumDriver class, open the Type Hierarchy. Notice, the ChromeDriver and EdgeDriver classes are both under the ChromiumDriver class. That's because ChromiumDriver is the parent. However, ChromiumDriver is a child to the parent RemoteWebDriver.



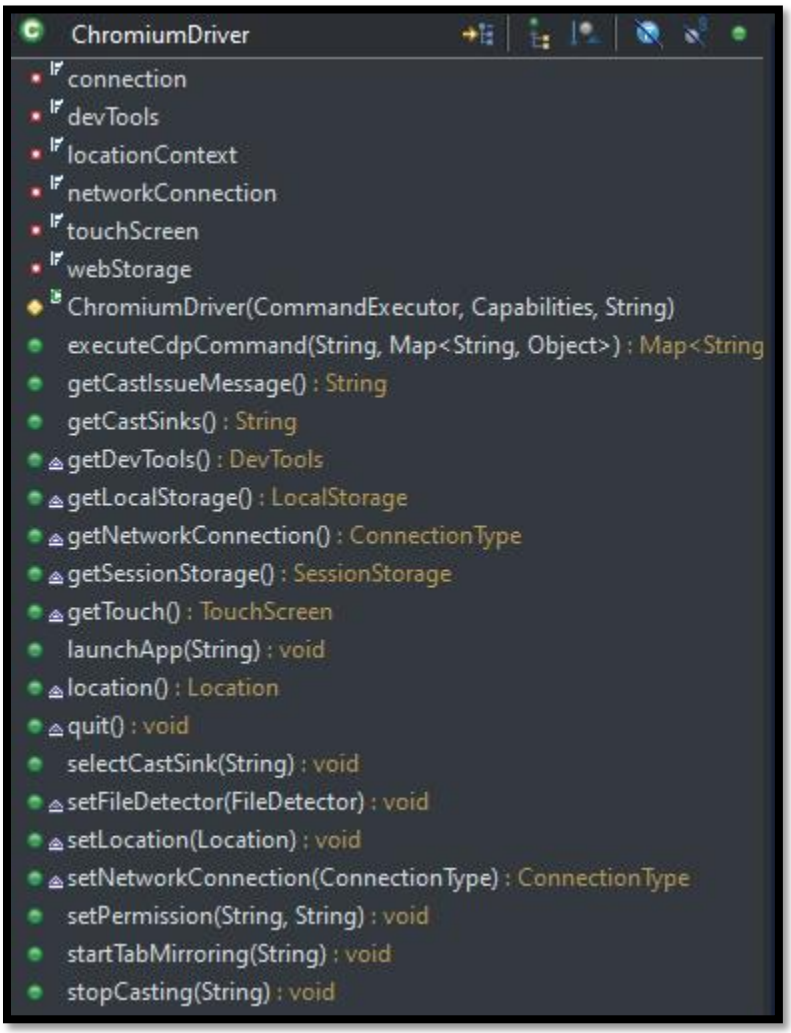
In a nutshell, EdgeDriver extends ChromiumDriver. ChromeDriver extends ChromiumDriver but ChromiumDriver extends RemoteWebDriver.

```
public class EdgeDriver extends ChromiumDriver {

 public EdgeDriver() { this(new EdgeOptions()); }

 public EdgeDriver(EdgeOptions options) {
  this(new EdgeDriverService.Builder().build(), options);
 }
```

```
public class ChromeDriver extends ChromiumDriver {

 /**
  * Creates a new ChromeDriver using the {@link ChromeDri
  * server configuration.
  *
  * @see #ChromeDriver(ChromeDriverService, ChromeOptic
  */
```

```
public class ChromiumDriver extends RemoteWebDriver
  implements HasDevTools, HasTouchScreen, LocationContext, Networ

 private final RemoteLocationContext locationContext;
 private final RemoteWebStorage webStorage;
 private final TouchScreen touchScreen;
```

There's a lot of methods in the ChromiumDriver class but 2 methods allow us to control Developer Tools in Chrome and Microsoft Edge.

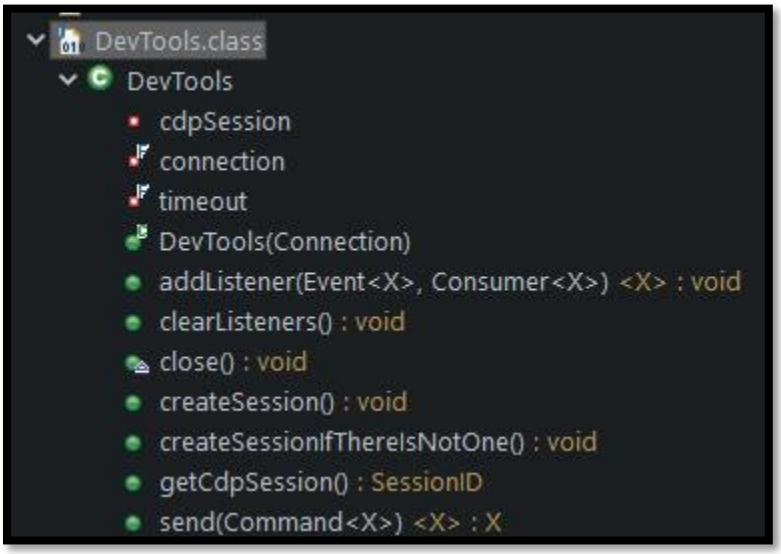


Those 2 methods are executeCdpCommand and getDevTools. The executeCdpCommand allows us to directly execute a Chrome DevTool Protocol command by passing in a parameter for that command. getDevTools is a method that returns DevTools.

```
public Map<String, Object> executeCdpCommand(String commandName, Map<String, Object> parameters) {
  Require.nonNull("Command name", commandName);
  Require.nonNull("Parameters", parameters);
```

```
@Override
public DevTools getDevTools() {
  return devTools.orElseThrow(() -> new WebDriverException("Unable to create DevTools connection"));
}
```

DevTools is a class that has methods to handle developer options. We see close, send, addListener. Know what, let me go back to the Project Explorer and look at the methods for DevTools. Here are the methods within DevTools: addListener, clearListener, close, createSession, createSessionIfThereIsNotOne, getCdpSession, and send.



We see a lot of methods for DevTools because we can do a lot of things with CDP. I'm going to show you how to view the Console Logs when it comes to CDP, Mock a GeoLocation, and Enable the Network Speed. Thanks for watching and I'll see you in the next session starting with View Console Logs. If you are interested in more videos, feel free to subscribe to my YouTube channel and click the bell icon. Also, follow me on Twitter, connect with me on LinkedIn and Facebook.

View Console Logs

There are 2 objectives for viewing a browser's Console panel. First, is to execute JavaScript. Second, is to view messages logged by the browser or logged by the developer. In this session, our focus will be to view messages logged by the browser. Most of the times, we view the Console Logs to get details about an error. With those details, we can identify the root cause of the problem.

Our AUT is this lego page and CTRL + SHIFT + I is a shortcut to the Developer Tools panel. In the Console tab, we already have some logs because the page has already been loaded. We can clear the logs, reload the page, then watch the logs back show up. DevTool failed to load SourceMap and HTTP error Status Code 404. The DevTools class in Selenium 4 provides a way to listen to these logs which has 4 Log Levels: Verbose, Info, Warnings, and Errors. Verbose is not checked for default. Therefore, the messages will not be wordy. It won't have many words in the message. Info means the logs will have important information about an event. Warnings indicate a strange condition might cause a problem with operations. Errors let us know a condition is likely to cause a problem with operations.

Now for our Test Script, chromedriver is setup and the window is maximized.

```
public class ViewConsoleLogs {

  ChromeDriver driver;

  @BeforeClass
  public void setUp () {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.manage().window().maximize();
  }
```

Start by writing @Test / public void viewConsoleLogs () { } Let's add some steps but write them as comments: First we // Get The DevTools and Create A Session. Then, // Send A Command To Enable The Logs. Next, we // Add A Listener For The Logs. Finally, we // Load The AUT

```
@Test
public void viewConsoleLogs () {
  // Get The DevTools And Create A Session

  // Send A Command To Enable The Logs

  // Add A Listener For The Logs

  // Load The AUT
}
```

To get the dev tools, we must cast the driver if WebDriver is our type. We cannot write driver.getDevTools. Notice, getDevTools is not available. We have to cast the driver by writing ((ChromeDriver) driver).getDevTools(). If you want to directly write driver.getDevTools, we must change the type to ChromeDriver. Now, I'm going to start over and write driver.getDevTools().

Do you see how getDevTools return DevTools? That's an indicator, this statement must be assigned to DevTools with an object reference of anyname but I'm going to write devTools = . Now, we have access to all of the methods I mentioned in the Introduction. Before, we perform any more steps, we must take control of the Developer Tools panel in the browser by creating a session: devTools.createSession();. The purpose of this statement is to start a session in the Chrome browser.

```
@Test
public void viewConsoleLogs () {
  // Get The DevTools And Create A Session
  DevTools devTools = driver.getDevTools();
  devTools.createSession();
```

Next, is to enable logs in the Console. devTools.send(). Send is a method that has built in commands and accepts Command as a parameter. It allows us interact with the Developer Tools. Log.enable() allows us to listen to the logs but for here we see Log.enable()in the next part is one of those commands for the send() method. Log serves as a representation or model for a CDP domain.

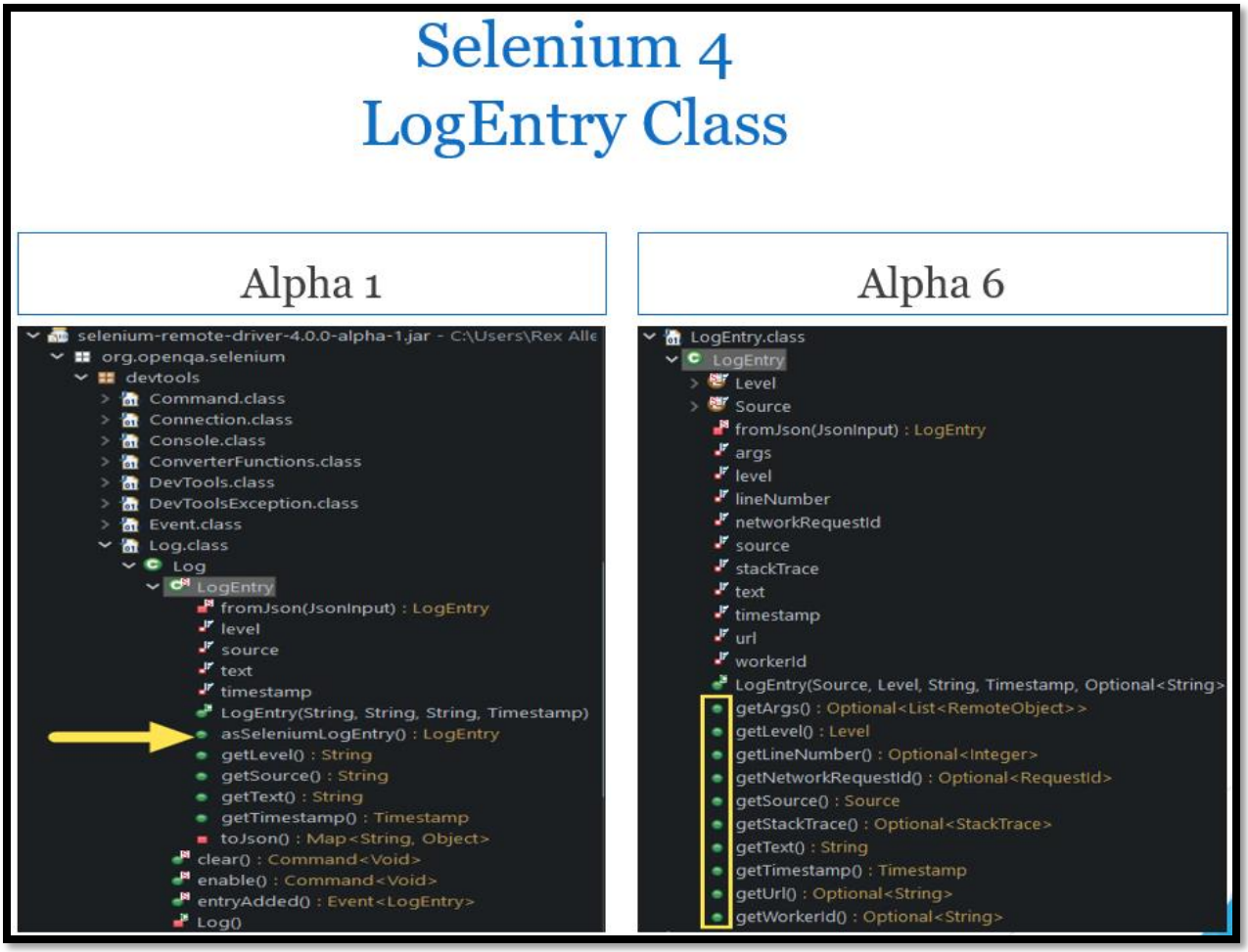
```
// Send A Command To Enable The Logs
devTools.send(Log.enable());
```

Now that we have the logs enabled, this is the part we go and listen for logs by writing devTools.addListener(). Once again, we write Log. but this time it's an event for entryAdded(), logEntry -> { }. To use this Lambda expression (->), you must have Java 1.8 or higher.

```
// Add A Listener For The Logs
devTools.addListener(Log.entryAdded(), logEntry -> {
  System.out.println(logEntry.asSeleniumLogEntry());
});
```

Let's print the log entries sout(logEntry.asSeleniumLogEntry()); This is where I noticed a difference between Selenium 4 Alpha 1 and the current version. I changed my pom.xml file to use the version for Alpha . Alpha 1 has asSeleniumLogEntry.

In this screenshot, we see Alpha 6 does not have asSeleniumLogEntry for a method. Alpha 6 is not the most recent version. However, Alpha 6 and the most recent version added more methods although asSeleniumLogEntry is missing. Alpha 1 does not have methods getArgs, getLineNumber, getNetworkRequestId, getStackTrace, getUrl and getWorkerId.



Here's my pom.xml file that shows Alpha 1.

```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0-alpha-1</version>
</dependency>
```

I'm going to add some hyphens then print the next Log Entries. sout("----------");
sout("source = " + logEntry.getSource());

Copy (“source = ” + logEntry.getSource()) then get the level getLevel() and change source to level Change level to text and getLevel to getText then level to timestamp and getLevel to getTimestamp.

```
// Add A Listener For The Logs
devTools.addListener(Log.entryAdded(), logEntry -> {
  System.out.println(logEntry.asSeleniumLogEntry());
  System.out.println("----------");
  System.out.println("source = " + logEntry.getSource());
  System.out.println("level = " + logEntry.getLevel());
  System.out.println("text = " + logEntry.getText());
  System.out.println("timestamp = " + logEntry.getTimestamp());
});
```

Last but not least we are going to load the AUT to view the Console logs is driver.get(); The URL is https://www.lego.com/404. Let’s Run. In the Console, we see some of the same information because that method asSeleniumLogEntry combines different log entries and we also printed individual log entries. For example, it has source as network, level = error, text = Failed to load resource and the timestamp.

```
Seen: {method=Log.entryAdded, params={entry={source=network, level=error,
 text=Failed to load resource: the server responded with a status of 404 (),
 timestamp=1.606829427178035E12, url=https://www.lego.com/en-us/404,
 networkRequestId=62F0CB1492417E91F6660123A45C65ED}},
 sessionId=180CE51405B2BC80F4ACCED20A964098}
[2020-12-01T07:30:27-0600] [SEVERE] Failed to load resource: the server responded
 with a status of 404 ()
----------
source = network
level = error
text = Failed to load resource: the server responded with a status of 404 ()
timestamp = 1606829427178
```

That’s it for capturing logs using Selenium 4 CDP and I’ll see you in the next session. If you are interested in more videos, feel free to subscribe to my YouTube channel and click the bell icon. Also, follow me on Twitter, connect with me on LinkedIn. The transcript and code will get placed on GitHub.

GeoLocation

In this session, I will show you how to mock a Geolocation using Selenium 4 CDP. Mock means imitate and Geolocation refers to the geographical location of a device connected to the internet. The device can be anything a phone, laptop, or even a watch. We use geolocation for events like finding the location of a place.

On Selenium's site, we see Chrome DevTools, Emulate Geo Location, description and code to Emulate the Geo Location.

```
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.devtools.DevTools;

public void geoLocationTest(){
  ChromeDriver driver = new ChromeDriver();
  Map coordinates = new HashMap()
  {{
      put("latitude", 50.2334);
      put("longitude", 0.2334);
      put("accuracy", 1);
  }};
  driver.executeCdpCommand("Emulation.setGeolocationOverride", coordinates);
  driver.get("<your site url>");
}
```

The description says "Some applications have different features and functionalities across different locations. But with the help of DevTools, we can easily emulate them." Depending on the application, we can see different information because the application is in a different location. With this help of the Chrome DevTools Protocol, the application can have the same information after mocking the geolocation. We can mock, well this site says emulate. We can emulate the geolocation by making our browser be in the same location. As a result, the application will have the same features and functionalities we are trying to test. For example, I live in Texas but on my project, I work with people who live in India. If our application showed different information in India then I would change my location to be in India. For this test, I'm going to make my browser be located in the capital of India. In this pom.xml file, I changed my Selenium 4 version to alpha 7.

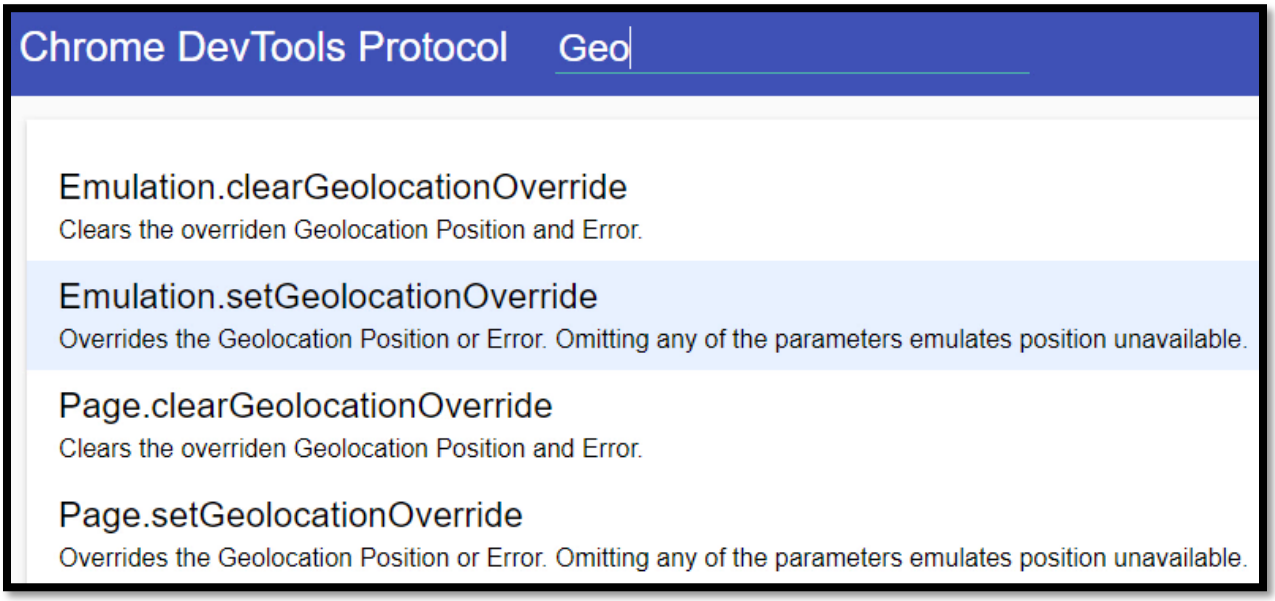
```
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.0.0-alpha-7</version>
</dependency>
```

From scratch, I'm going to write @Test / public void mockGeoLocation () { } set up our browser by writing WebDriverManager.chromedriver().setup();ChromeDriver is the type and driver = is the object

new ChromeDriver ();. Maximize the window: driver.manage().window().maximize(); / driver.exec
.Notice, how executeCdpCommand has String commandName and Map for the parameters.

```
@Test
public void mockGeolocation () {
  WebDriverManager.chromedriver().setup();
  ChromeDriver driver = new ChromeDriver();
  driver.manage().window().maximize();
  driver.executec
}       executeCdpCommand(String commandName, Map<String, Object>
        Ctrl+Down and Ctrl+Up will move caret down and up in the editor  Next Tip
```

In the sample code, the command is Emulation.setGeolocationOverride and coordinates for the Map. The coordinates are latitude, longitude, and accuracy. For more information, we can go to github. The url is https://chromedevtools.github.io/devtools-protocol/, search for Geo and we see Emulation.setGeolocationOverride with a description that says Overrides the Geolocation Position or Error. We see all 3 parameters: latitude, longitude, and accuracy.



Emulation.setGeolocationOverride #

Overrides the Geolocation Position or Error. Omitting any of the parameters emulates position unavailable.

PARAMETERS

latitude	optional	**number** Mock latitude
longitude	optional	**number** Mock longitude
accuracy	optional	**number** Mock accuracy

I'm going to set my geolocation to be at New Delhi. We see the coordinates are 28.6139 for latitude and 77.2090 for longitude. Latitude is always first. We see N for Latitude because the direction is North or South. E for Longitude because the direction is East or West. Our application under test is Where Am I Right Now. I guess this site is accurate. The location is half a mile from my house. We see TX for Texas.

Now, let's go ahead and complete our script. 1st parameter is String commandName. We saw Emulation.setGeolocationOverride. The 2nd parameter was a Map and copy this code from Selenium's site and paste it to our code. Change latitude to 28.6139 and longitude is 77.2090, leave accuracy at 1. Now, I can pass in coordinates. Last step, is to load the Application Under Test: driver.get("https://where-am-i.org/");

```
Map coordinates = new HashMap()
{{
  put("latitude", 28.6139);
  put("longitude", 77.2090);
  put("accuracy", 1);
}};
driver.executeCdpCommand(
        commandName: "Emulation.setGeolocationOverride",coordinates);
driver.get("https://where-am-i.org/");
```

Now, let's Run. We see New Delhi for Location. We see Latitude and Longitude. Also, here's the map.

Your Location

Rajpath, Central Secretariat, New Delhi, 110001, Delhi, India

Your Latitude and Longitude

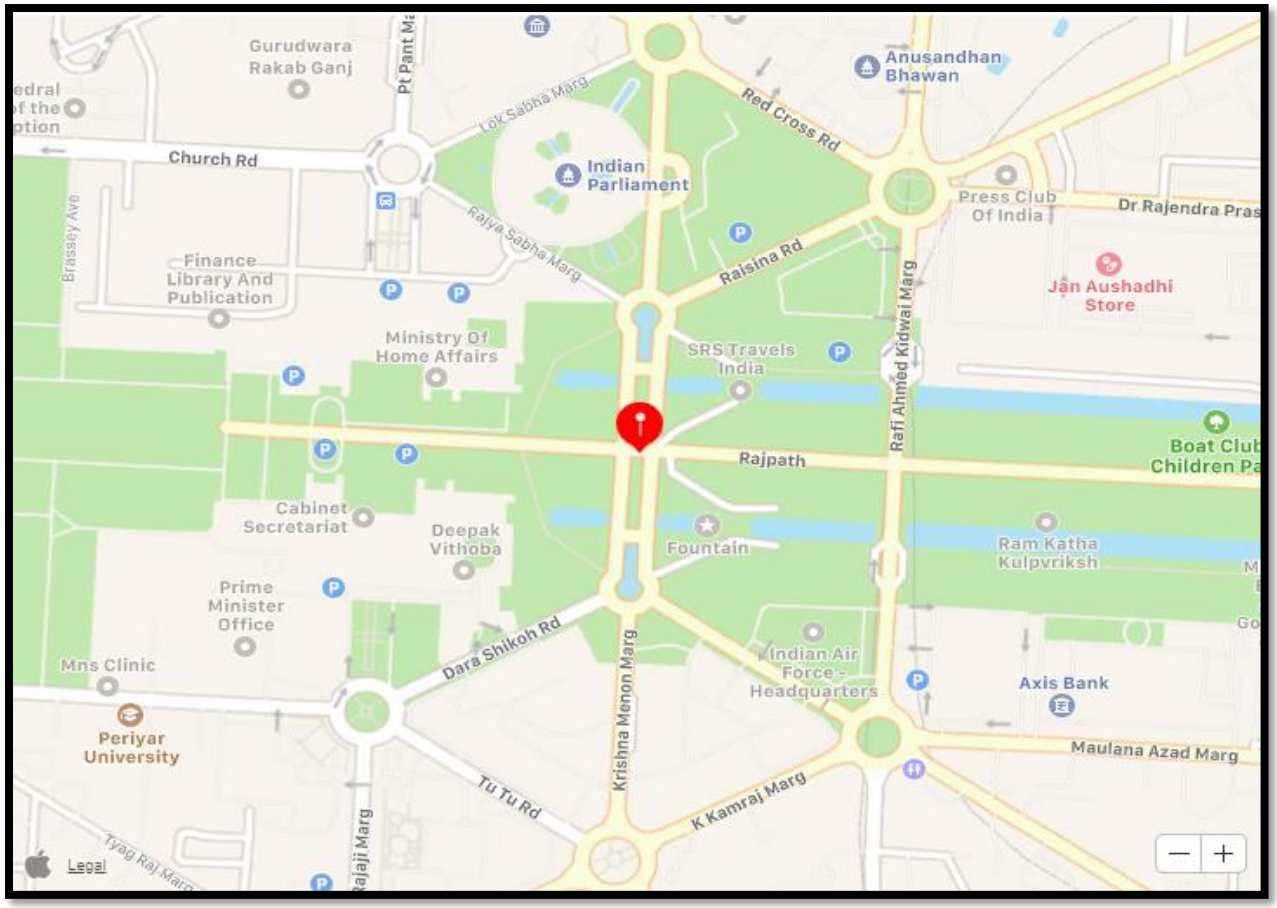
28.6139

77.209

DMS (degrees, minutes, seconds)

N 28 ° 36 ' 50.04 "

E 77 ° 12 ' 32.4 "



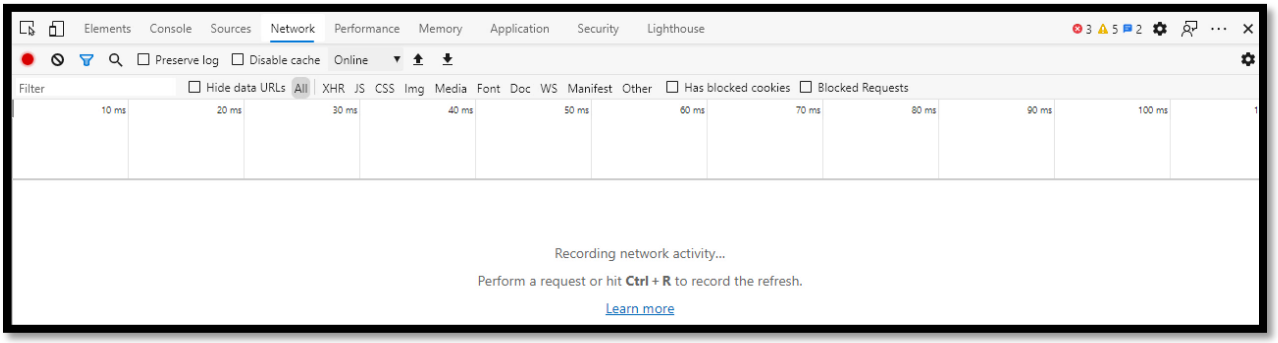
Thanks for watching and I'll see you in the next session for enabling the network.

Enable Network

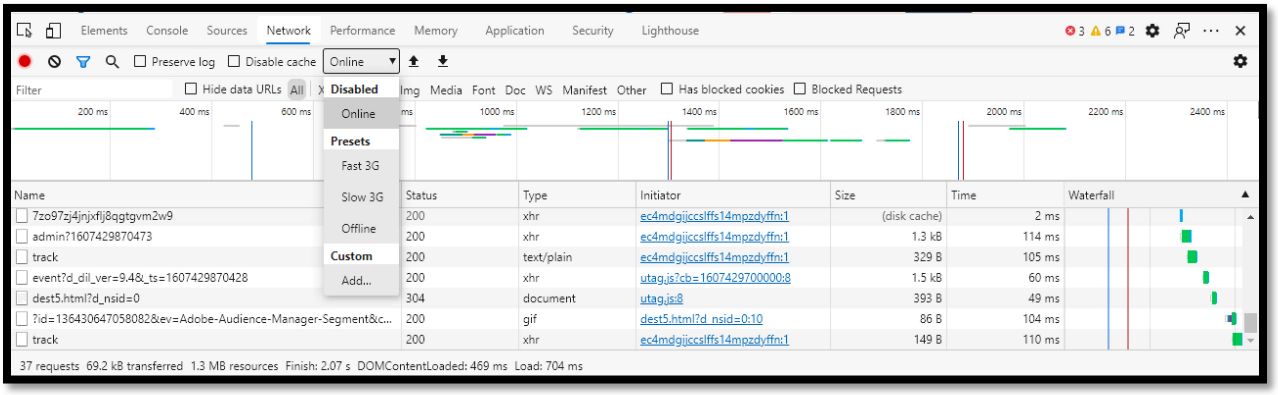
The Network panel is another way to debug a problem. It has logs to help us troubleshoot the problem. We mostly use the panel for 2 reasons. One reason is to inspect the network properties of a resource. Another reason is to make sure our resources are downloaded and uploaded as expected. When testing an application, it's easy to forget or not think about a user with a weak connection. Therefore, we are going to automate how to slow down our internet connection.

Slow Connection

In the Network panel, there's no available request. It's empty because we must open the panel before performing an action. That's how we preserve the network traffic data.



Refresh the page and all of the network activity shows up in the Network Log. Each row in the Network Log represents a resource. Do you see 69.2 kB transferred? That's the total download size. This Network Throttle dropdown is how we emulate different connection speeds. We see Fast 3G, Slow 3G, and Offline.



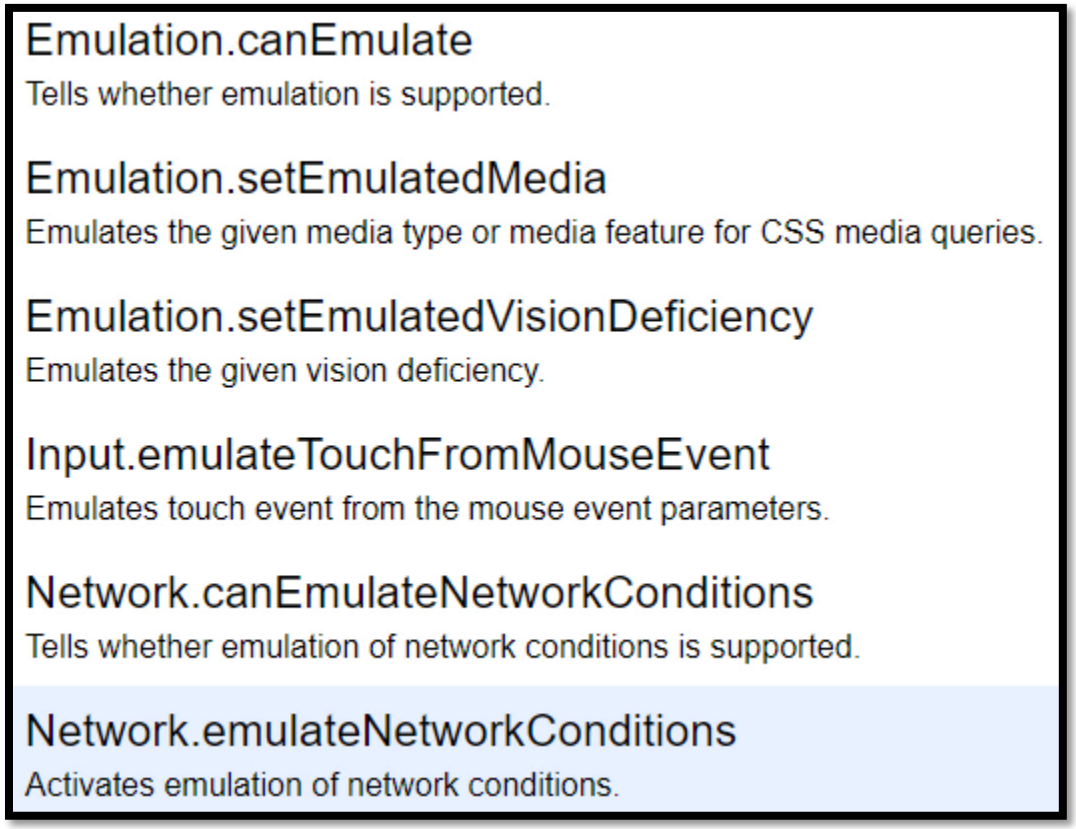
In this session, I will automate a 3G connection but next session I will take the network offline. Select any preset and the DevTools display a warning icon beside the Network tab.

For our Test Script, I have the setup method for EdgeDriver. Let me also add driver.getDevTools() assign it to devTools = . Also write DevTools devTools; up top.

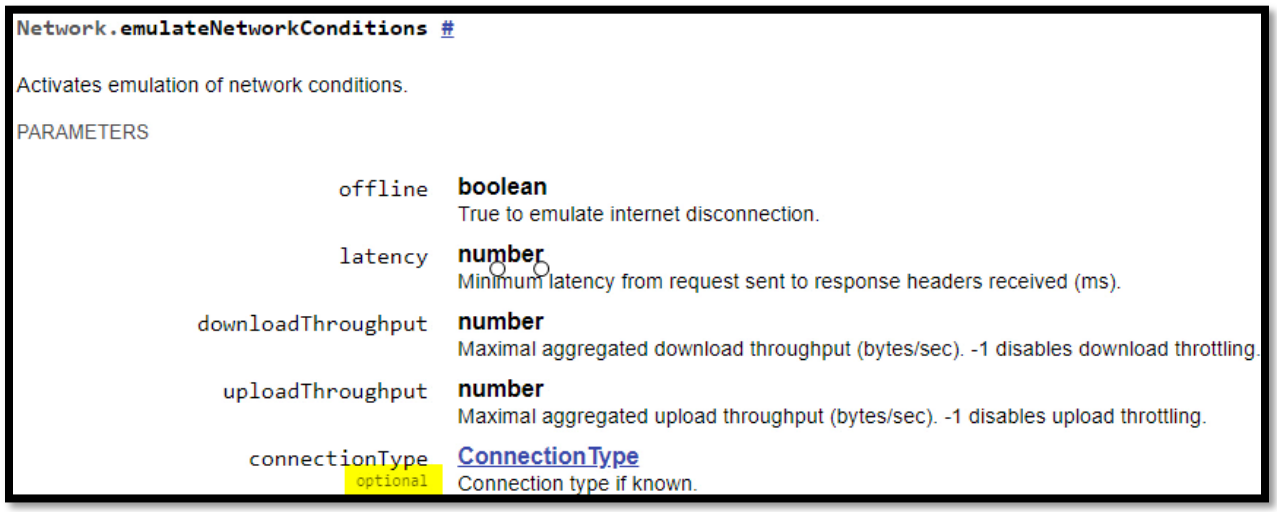
```
public class EnableNetworks {

  EdgeDriver driver;
  DevTools devTools;

  @BeforeMethod
  public void setUp () {
    WebDriverManager.edgedriver().setup();
    driver = new EdgeDriver();
    driver.manage().window().maximize();
    devTools = driver.getDevTools();
  }
```

Enable the network to slow down by writing. @Test public void enableSlowNetwork () { } Start by creating a session devTools.createSession(); then we send a command to enable the network by writing devTools.send(Network.enable(Optional.empty(), let's write this statement 2 more times. This complete statement makes it possible to deliver network tracking and events to the client. All of the parameters are Optional.empty because they are not required. In fact, the first 2 parameters are experiments.

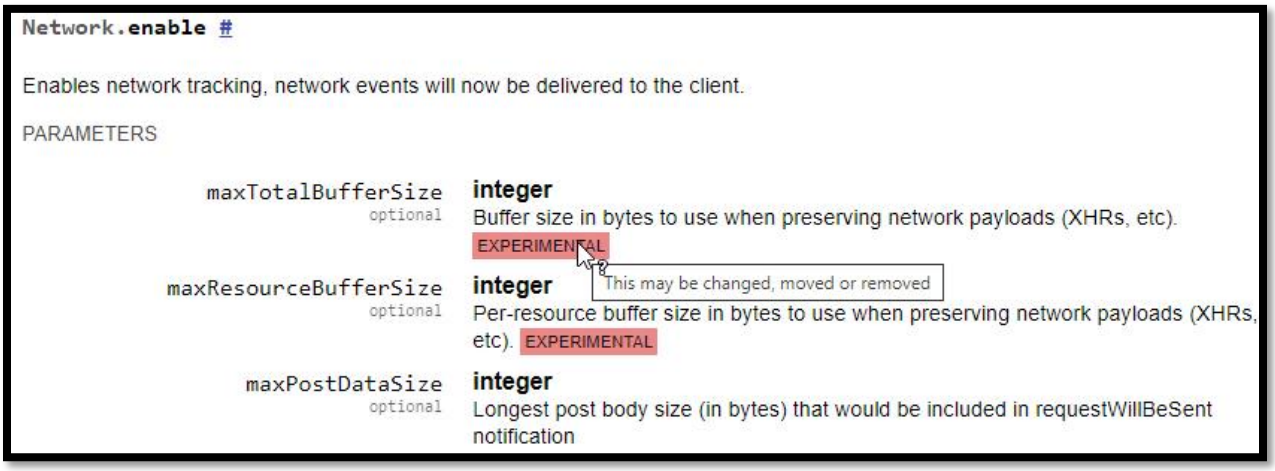
Next, is to emulate the Network Conditions. We can go to github for the method and parameters. Search for emulate. At the bottom is Network.emulateNetworkConditions. It activates emulation of network condition.



The parameters are offline, latency, downloadThroughput, uploadThroughput, and connectionType. Connection Type is the only optional parameter.



The method we wrote for enabling the network is right here and we see all 3 parameters are optional: maxTotalBufferSIze, maxResourceBufferSize, and maxPostDataSize. If I hover Experimental, the tool tip says “This may be changed, moved, or removed”.



For our Test Script, let's complete it by writing devTools.send(Network.emulateNetworkConditions(). We want the network to stay online so we set offline to false, latency is 150, downloadThroughput is 2500, uploadThroughput is 2000. Next, is the Connection Type. Optional.of(ConnectionType.) We see different choices. We have BLUETOOTH, 2G, 3G, 4G, and WIFI. Let's go ahead and select 3G.

Let's also compare the time between this slow network and the normal way by loading an application. Know what before we compare, we must load the application for LinkedIn: driver.get("https://www.linkedin.com"). Let's also print the page title. sout("Slow " + driver.getTitle).

```
@Test
public void enableSlowNetwork () {
  devTools.createSession();
  devTools.send(Network.enable(
          Optional.empty(),
          Optional.empty(),
          Optional.empty()));
  devTools.send(Network.emulateNetworkConditions(
           offline: false,
           latency: 150,
           downloadThroughput: 2500,
           uploadThroughput: 2000,
          Optional.of(ConnectionType.CELLULAR3G)));
  driver.get("https://www.linkedin.com");
  System.out.println("Slow " + driver.getTitle());
}
```

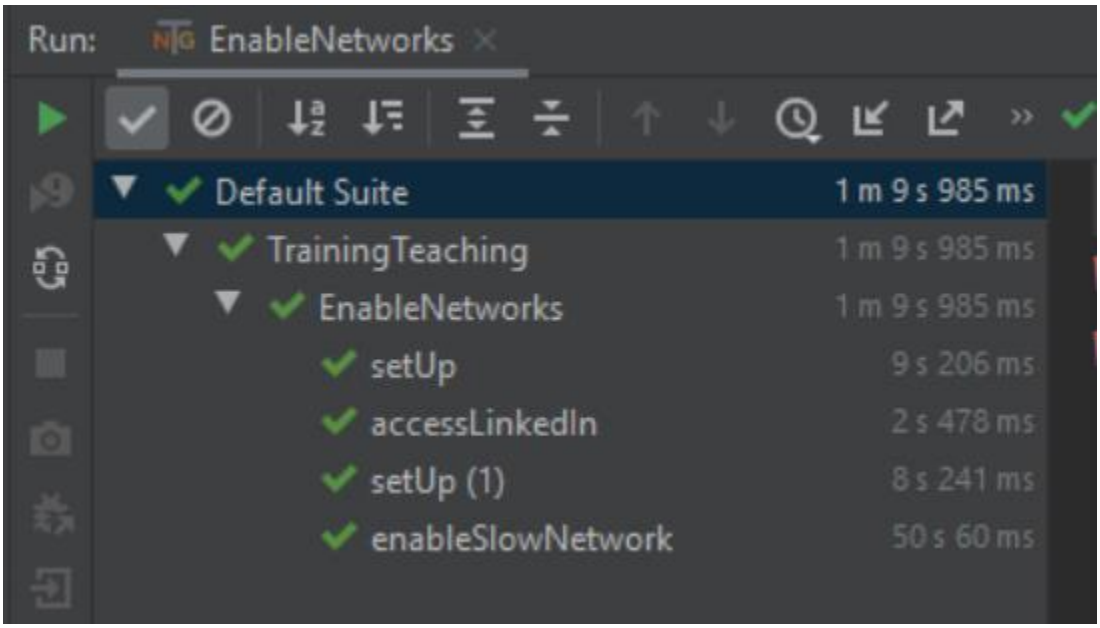
Now, let's go ahead and load LinkedIn the normal way.

@Test public void accessLinkedIn () { } driver.get("https://www.linkedin.com"). Also sout("Access " + driver.getTitle());

```
@Test
public void accessLinkedIn () {
  driver.get("https://www.linkedin.com");
  System.out.println("Access " + driver.getTitle());
}
```

Let's go ahead and run.

The Console shows accessLinkedIn loaded in 2 Seconds and 478 Milliseconds. enableSlowNetwork took longer to complete. It finished in 50 Seconds and 60 Milliseconds. Both Test Scripts show the Page Title. That's it for slowing down the Network to a slower connection.

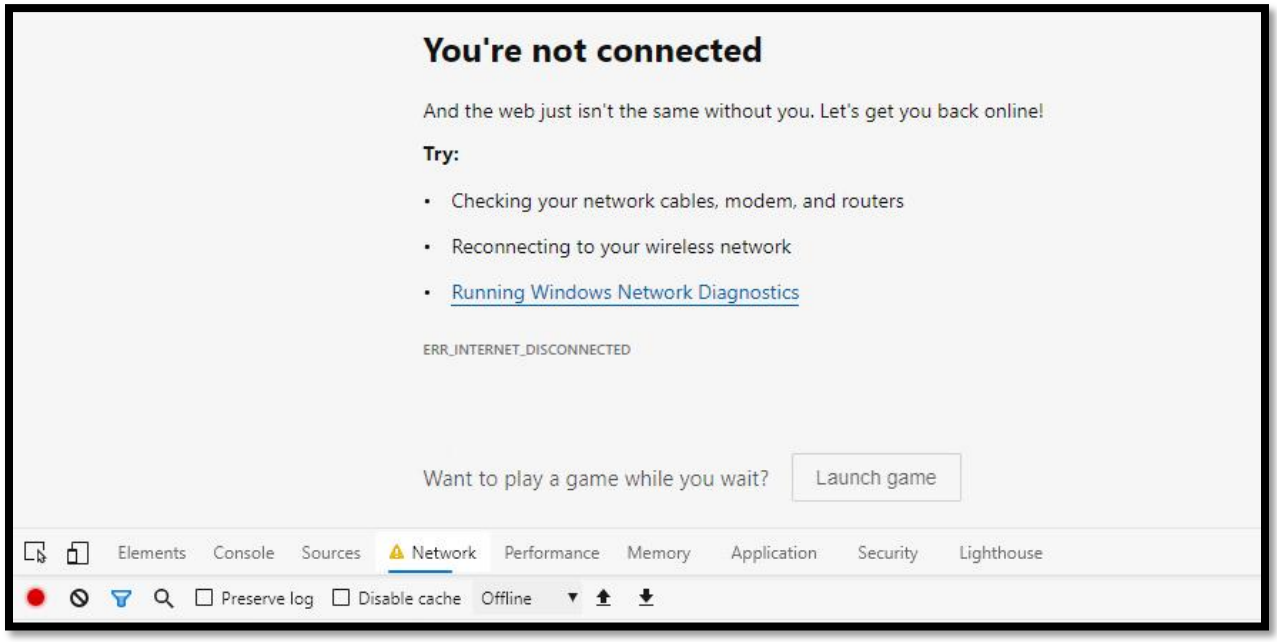


Next, I'm going to show you have to take the Network offline. If you are interested in more videos, consider subscribing to my YouTube channel and clicking the bell icon. Also, follow me on Twitter, connect with me on LinkedIn and Facebook. The transcript and code will get placed on GitHub. Thanks for watching and I'll see you in the next session.

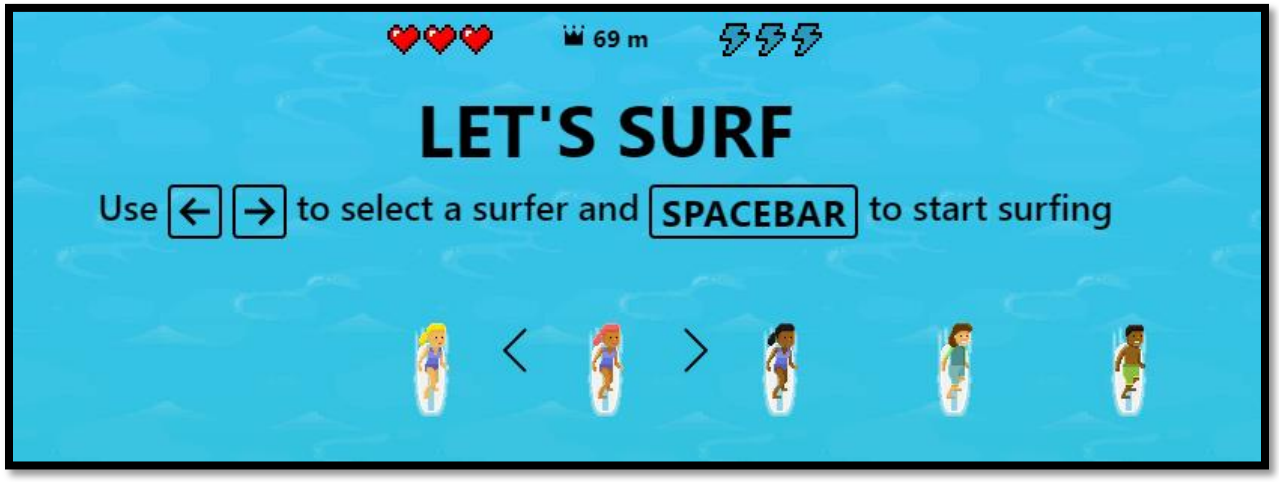
Emulate Offline

In this session, we are going to enable the network to be offline. It's very good when an application has functionality to continue working offline. There are many reasons for an application to be offline such as bad weather, a power outage, or even driving into a dead zone.

Here's our Application Under Test. Most of the times, I use Chrome but not this time. For some reason Chrome does not include the same offline functionality as Microsoft Edge. I'm going to right click, Inspect, and go to the Network tab. The Network Throttling drop down has different options. We want Offline. Refresh and notice the message says "You're not connected" and it also shows INTERNET_DISCONNECTED. A lot of applications stop right here and do not let us continue. However, we can still perform an action while offline. Do you see Want to play a game while you wait and this Launch game button?



Inspect this launch game button and we see the id value is game-button. Now, at this point, we are going to do the same steps in our automation. In our test, we are going to set the test to be offline then click this Launch game button. It's a surf game.

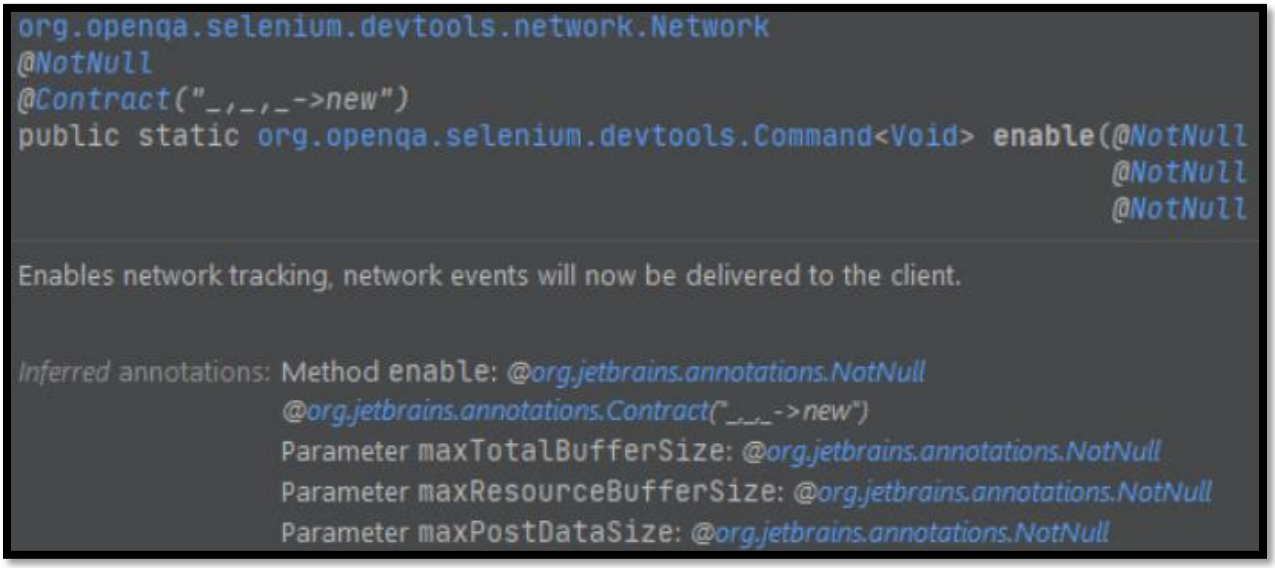


For our Test Script, I already have EdgeDriver, DevTools, and the setup method for EdgeDriver. It's from the last session when we slowed down the network connection using 3G and accessed LinkedIn.

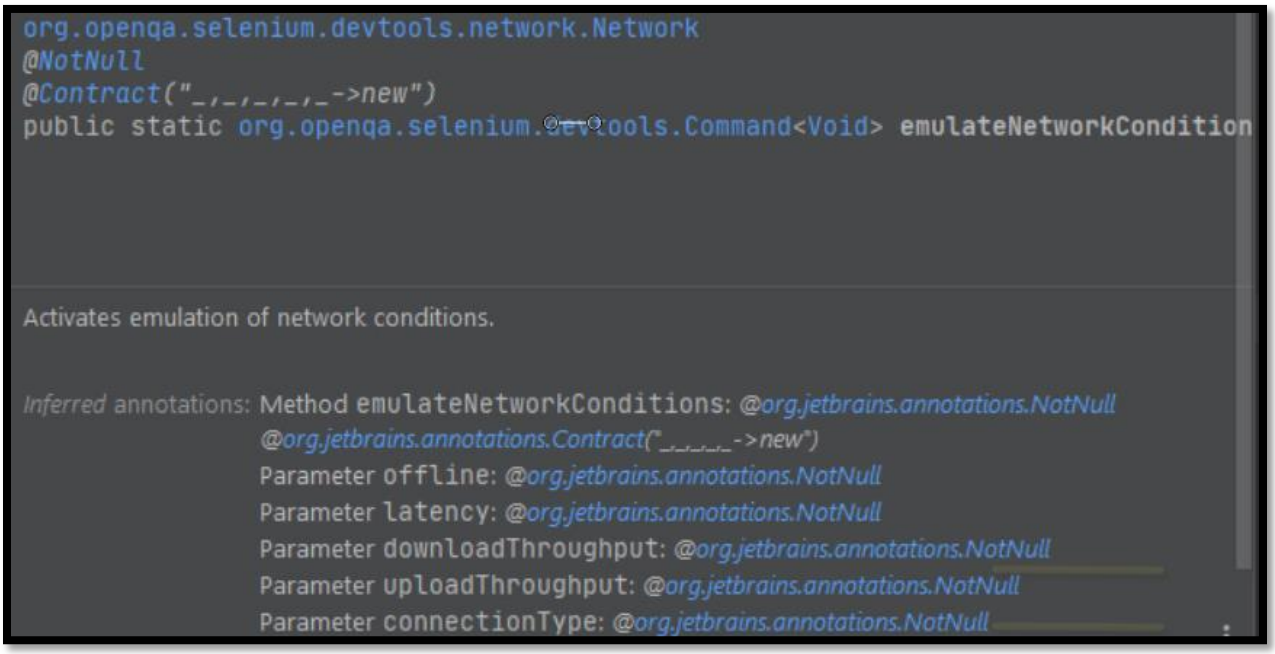
Now, let's write @Test public void enableOfflineNetwork () { }. We always begin by creating the session. devTools.createSession(). This will start a session in the Edge browser. Next, is to send a command so we can enable the network. devTools.send(Network.enable()) The enable method has 3 parameters and all 3 of them are optional. Therefore, we write Optional.empty(), Optional.empty(), Optional.empty().

```
@Test
public void enableOfflineNetwork () {
  devTools.createSession();
  devTools.send(Network.enable(
          Optional.empty(),
          Optional.empty(),
          Optional.empty()));
```

If I hover enable, the description shows "Enables network tracking, network events will now be delivered to the client". Those 3 empty parameters are maxTotalBufferSize, maxResourceBufferSize, and maxPostDataSize.



After enabling the network, we are going to send a command to emulate the network. devTools.send(Network.emulateNetworkConditions()). Hover the method to see it activates an emulation of the network conditions using 5 parameters: offline, latency, downloadThroughput, uploadThroughput and connectionType. I don't see it here but Connection Type is optional.



Now, let's go ahead and continue with our test script by writing the value for offline and that value will be true. This means yes, the network will be offline. 10 is the value for latency. Latency is sometimes called lag. It's a delay in communication across the network. 100 for the downloadThroughput and 50 for the uploadThroughput. Throughput refers to the amount of data that transfers from the source to

the destination. For connection type, we write Optional.of(ConnectionType.G) and we see the different G connections but let's select WIFI.

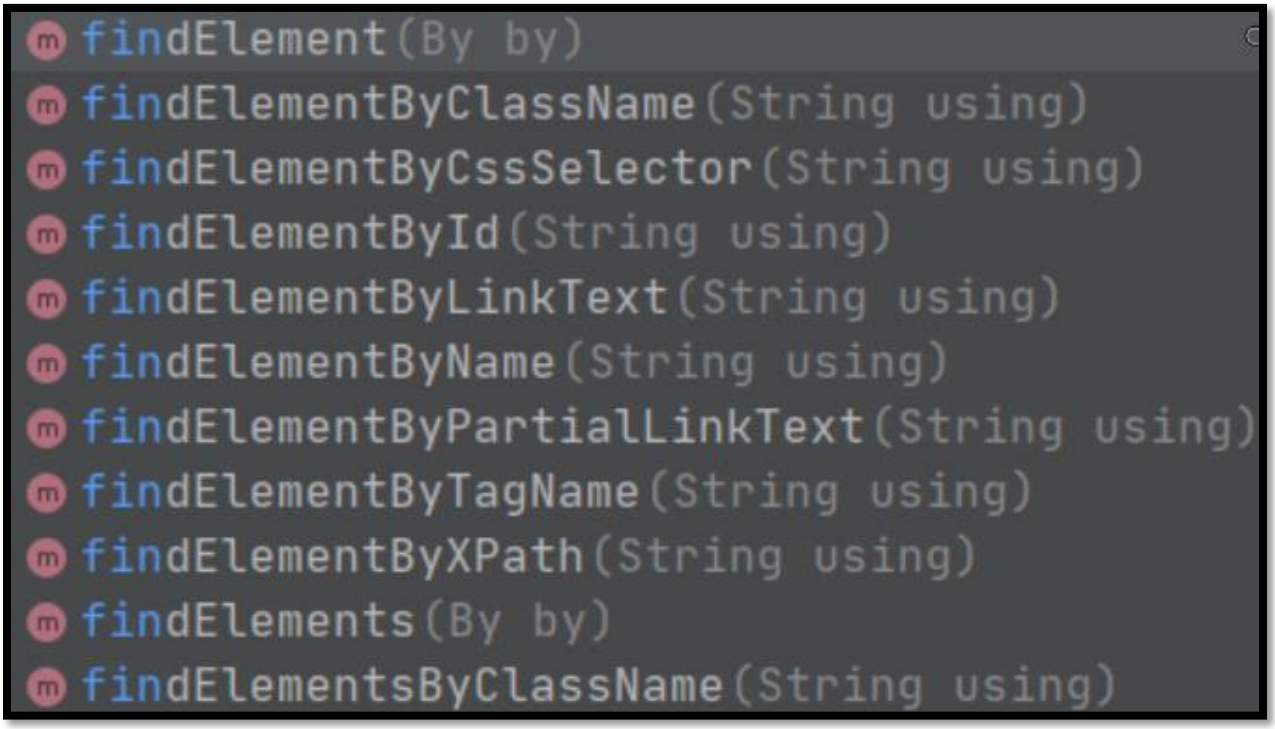
```
devTools.send(Network.emulateNetworkConditions(
        offline: true,
        latency: 10,
        downloadThroughput: 100,
        uploadThroughput: 50,
       Optional.of(ConnectionType.WIFI)));
```

Now, let's add a listener and what we expect. devTools.addListener() At this point, I'm going to write some data inside the addListener some parameters and that starts with (loadingFailed(),) and we are going to select the example with lambda expression because there are 2 options. Now, we write assertEquals() and inside assertEquals we are going to write loadingFailed and select the one that do not have parenthesis dot getErrorText(). Remember when we saw Internet Disconnected after setting the network to offline then refreshing the page. That's what we expect "net::ERR_INTERNET_DISCONNECTED"));

```
devTools.addListener(loadingFailed(),
       loadingFailed -> assertEquals(loadingFailed.getErrorText(),
               expected: "net::ERR_INTERNET_DISCONNECTED"));
```

The last step is to load the page. Since it's offline, let's try { } to load the page by writing driver.get("https://www.google.com").

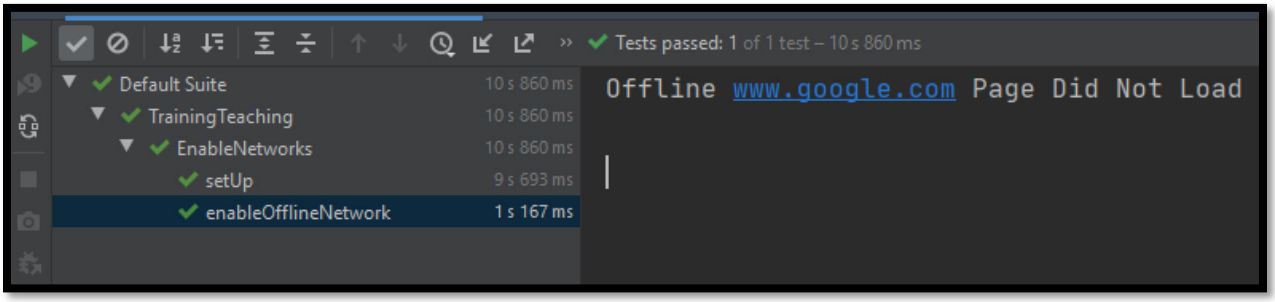
Next, is to catch () the possible exception (WebDriverException exc) { } and click the button by writing driver.find. Do you see the extra findElementBy options? We have these options because our type is EdgeDriver. It's been available to us for a long time. The same with ChromeDriver and FirefoxDriver. However, they supposed to be deprecated for Selenium 4.



I spoke about this in the Introduction to Selenium 4. The first video. I'm going to select driver.findElement(By.id("game-button")).click(). Also, print the page title: sout("Offline " + driver.getTitle() + "Page Did Not Load");

```
try {
  driver.get("https://www.google.com");
} catch (WebDriverException exc) {
  driver.findElement(By.id("game-button")).click();
  System.out.println("Offline " + driver.getTitle() + " Page Did Not Load");
}
```

That's it and let's run. It passed and we see the game. Now, for our Console, let's see what it shows enableOfflineNetwork. We see it shows Offline www.google.com and the Page Did Not Load.

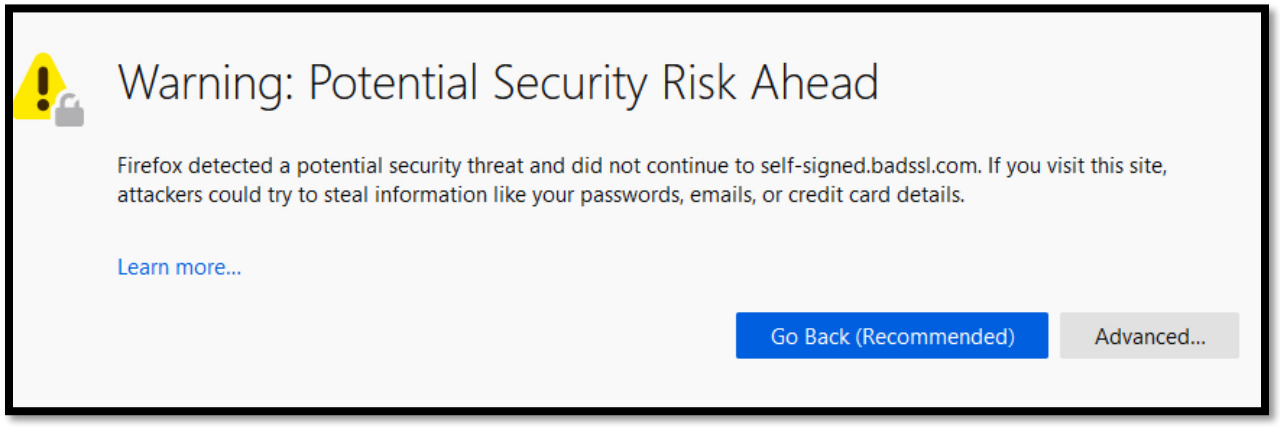


Thank You for watching and I'll see you in the next session. I create videos every week and if you are interested in videos that I create, feel free to subscribe to my YouTube channel and click the bell icon. Also follow me on Twitter, connect with me on LinkedIn and Facebook. I will make sure to place the transcript and code on GitHub.

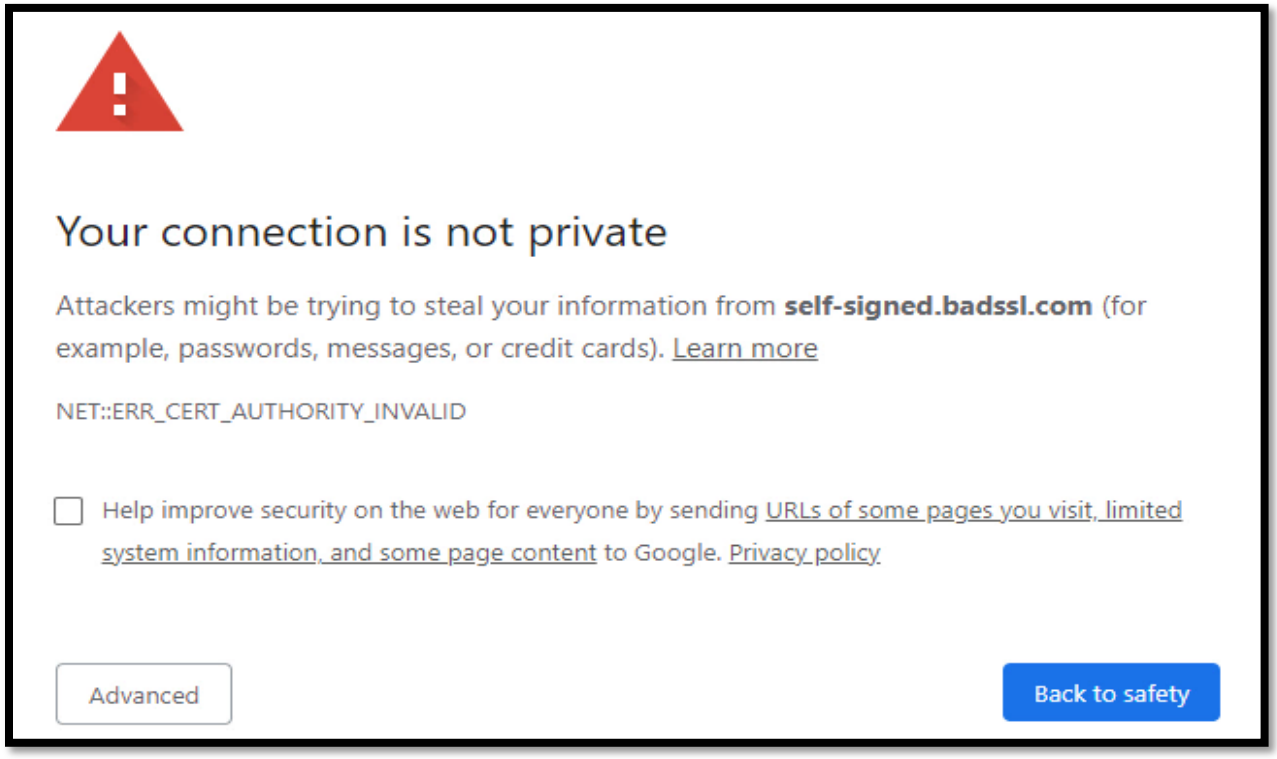
Certificate Error Website

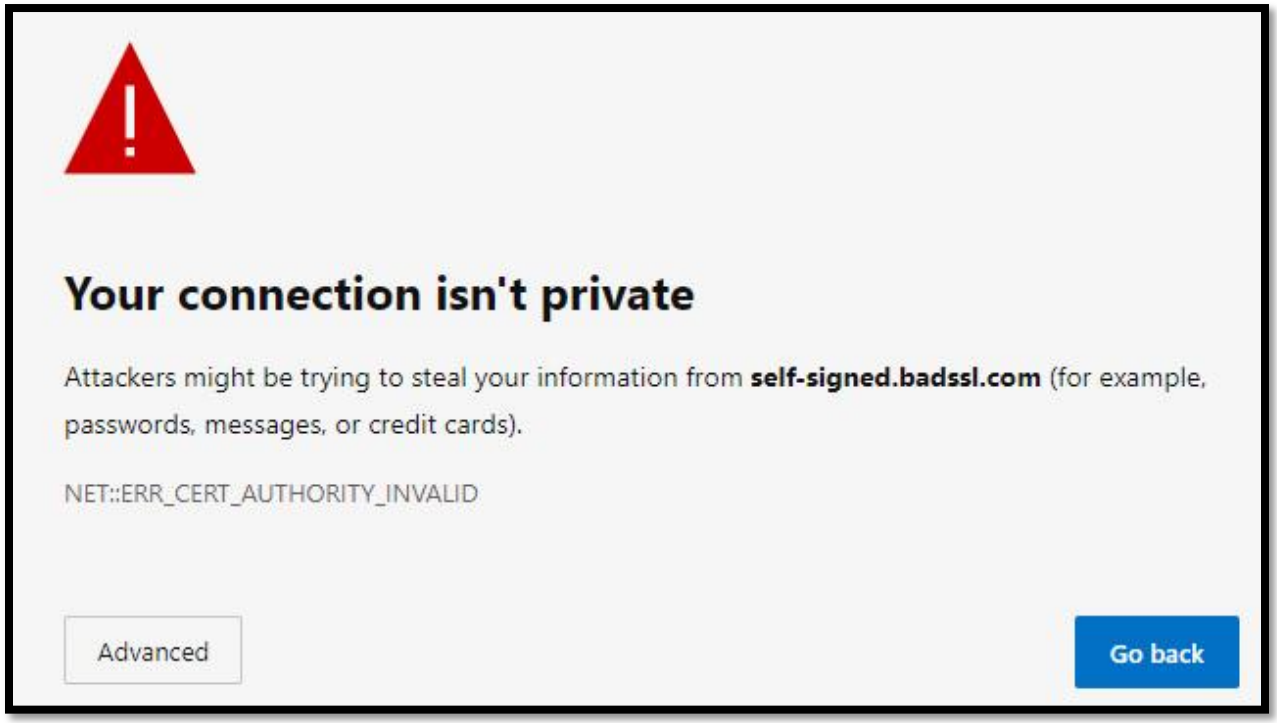
In this session, our focus will involve loading an untrusted website. Depending on your browser, there will be a different message but it all comes down a security issue. It's recommended we do not visit the site. However, there may be a scenario for us to test this type of application.

On Firefox, the message says "Warning: Potential Security Risk Ahead". It's letting us know that an attacker could try to steal our information.



Chrome has a similar message about attacker might be trying to steal your information. With a heading that says "Your connection is not private". Edge has the same message as Chrome.


Your connection is not private
Attackers might be trying to steal your information from self-signed.badssl.com (for example, passwords, messages, or credit cards). Learn more
NET::ERR_CERT_AUTHORITY_INVALID
Help improve security on the web for everyone by sending URLs of some pages you visit, limited system information, and some page content to Google. Privacy policy
Advanced
Back to safety


Your connection isn't private
Attackers might be trying to steal your information from self-signed.badssl.com (for example, passwords, messages, or credit cards).
NET::ERR_CERT_AUTHORITY_INVALID
Advanced
Go back

Our automation Test Script will ignore these security warnings and load the page. If I click the Advanced button, we see a link that says “Proceed to the self-signed.badssl.com (unsafe)”. Click this link and the self-signed.badssl.com site load without a problem.

GitHub has more example sites just like this one. We see 5 sites that have subdomains that generate the same type of warning

For our Test, its already been set up for ChromeDriver and DevTools. The last statement assigns getDevTools to devTools.

```
public class UntrustedWebsite {

  ChromeDriver driver;
  DevTools devTools;

  @BeforeMethod
  public void setUp () {
    WebDriverManager.chromedriver().setup();
    driver = new ChromeDriver();
    driver.manage().window().maximize();
    devTools = driver.getDevTools();
  }
```

Let's start by creating a method with @Test public void loadBadWebsite () { } Next, we create a session with devTools.createSession(). Now, we are going to send a command to ignore the certificate error.

devTools.send(Security.) Security provides access for us to setIgnoreCertificateErrors(). This statement is similar to clicking the Advanced button and the unsafe hyperlink. We pass in a boolean value of (true). True is another way of saying we trust the website. So, it will successfully load the page. That's it, last part is to load the AUT. driver.get("https://self-signed.badssl.com/");

```
@Test
public void loadBadWebsite () {
  devTools.createSession();
  devTools.send(Security.setIgnoreCertificateErrors(true));
  driver.get("https://self-signed.badssl.com");
}
```

The key is ignoring the Certificate Error by passing in true. A negative Test Case is to pass in false which will not load the untrusted website. Now, I'm going to paste the code for what we just used to ignore and change true to false. Change the name from loadBadWebsite to doNotLoadBadWebsite.

```
@Test
public void doNotLoadBadWebsite () {
    devTools.createSession();
    devTools.send(Security.setIgnoreCertificateErrors(false));
    driver.get("https://self-signed.badssl.com");
}
```

Let's Run. They both Passed. One browser shows self-signed.badssl.com and the other browser shows Your connection is not private. That's it.

Thanks for watching and I'll see you in the next session. If you are interested in more videos, feel free to subscribe to my YouTube channel and click the bell icon. Also, follow me on Twitter, connect with me on LinkedIn, and Facebook. The transcript and code will get placed on GitHub.

Contact Info

- ✔ YouTube https://www.youtube.com/c/RexJonesII/videos
- ✔ Facebook https://facebook.com/JonesRexII
- ✔ Twitter https://twitter.com/RexJonesII
- ✔ GitHub https://github.com/RexJonesII/Free-Videos
- ✔ LinkedIn https://www.linkedin.com/in/rexjones34/