Rex Jones II

(Transcript)
Selenium Actions

Table of Contents

Rex Jones II

Selenium Action vs Actions

Introduction

Hello and Welcome, My name is Rex Jones II. This video is an Introduction To The Actions Class In Selenium.

We are going to start with the difference between Action without an “s” and Actions with an “s”. Both are part of Selenium’s interaction package.

Action is an interface representing 1 user-interaction. It only has 1 method and that method is perform. The Actions class has a lot of methods like build, dragAndDrop, dragAndDropBy, keyDown, keyUp, moveToElement, and it also has perform. This perform() method is different from the one in the Action Interface. The description shows perform is a convenience method for performing the actions without calling build() first. The purpose of build() is to collect a sequence of actions and store all of those actions into 1 action. That’s why the description states “Generates a composite action containing all actions.”

Let’s walkthrough an example to help us see the difference between Selenium’s Action Interface and Actions Class.

Rex Jones II

Demo

We are going to enter all capital letters TEST into the Search box. Inspect the search box and we see q as the value for name. Now, let's go to Eclipse. In this method, we are going to enter the Actions Class and Action Interface. Find the element by writing WebElement searchBox = driver.findElement(By.*name*("q"));

Instantiate the Actions class and act as the object = new Actions. We need the driver in parenthesis to instantiate the class. Next, we write the sequence of actions to make all capital letters. Capital Letters are created when we press down the Caps Lock key or press down the Shift Key. Let's press down the Shift Key by writing act.keyDown. The WebElement is searchBox then Keys.SHIFT).

The next action is to enter text in the search box act.sendKeys. We can select anyone of the sendKeys methods. Let's select the one with WebElement. searchBox and the sequence is "with build". It does not matter if we send the keys with lowercase, mixed case, or uppercase letters. We keyed down the Shift Key, now we key up the Shift key. act.keyUp(searchBox, Keys.*SHIFT*);

Now, generate a composite of all 3 actions by building the actions. act.build() and assign it to Action interface. This is how we include the Action Interface. The last step is perform all of these actions by writing action.perform. Let's Run. We see WITH BUILD in all caps.

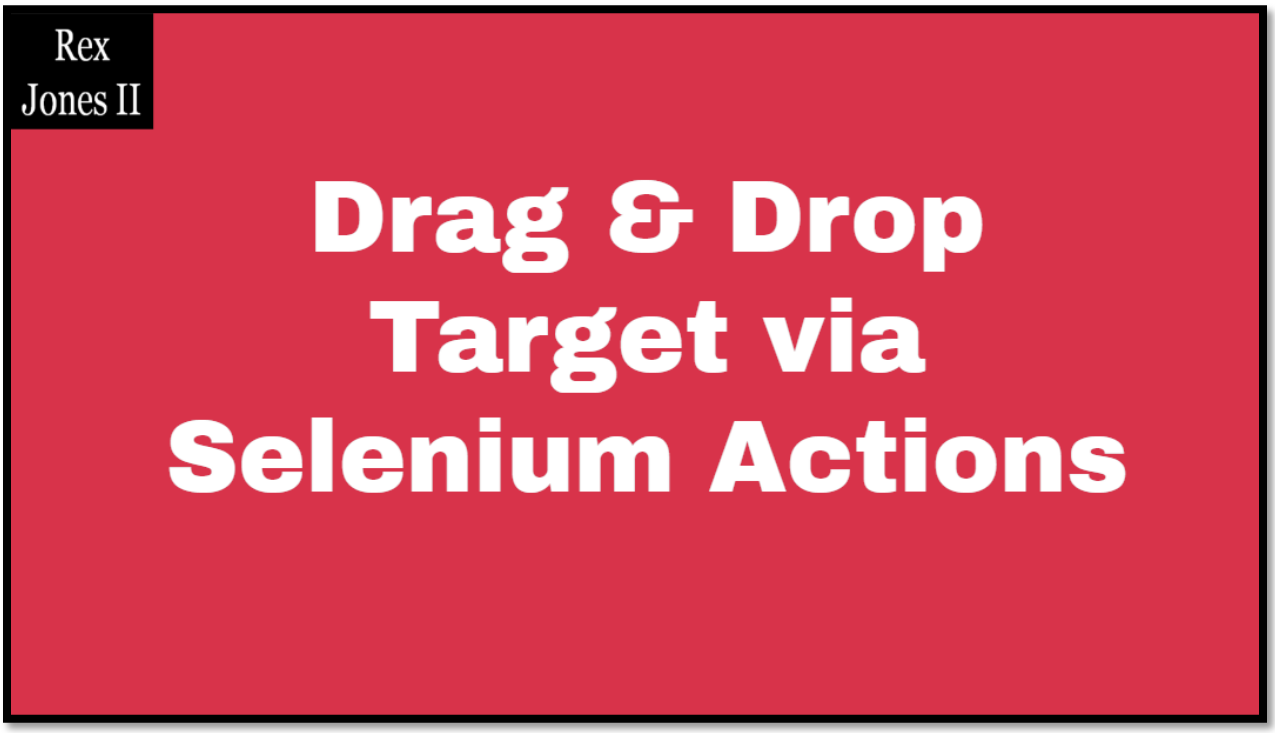
Recall perform in the Actions class. It said perform actions without calling build first. Now, let's execute these same actions without calling the build method. Copy, Paste, add s to Action, and change with to without. We don't need the build() method because perform() in the Actions class will automatically build the actions and perform the actions. Therefore, we write act.perform().

Now, we have 2 perform() methods but the one from the Action Interface is no longer necessary. Let's remove the build and perform methods. There's 1 more concept. All of these actions can be combined into 1 statement line.
act.keyDown(searchBox, Keys.***SHIFT***).sendKeys(" Works Too").keyUp(searchBox, Keys.***SHIFT***).perform();
Let's run. WITH BUILD / WITHOUT BUILD WORKS TOO

Thanks for watching Introduction to Selenium's Actions class – What Is The Difference Between Selenium Action Interface and Action Class. Next, we are going to drag and drop target.

Rex
Jones II

Drag & Drop
Target



In this video, we are going to drag this element to this Drop Here element using Selenium's Actions Class. Inspect both WebElements. Right click the first WebElement and we see draggable as the value for id. Right click the second element and the value for id is droppable.

Let's go to Eclipse, we have our set up and tear down methods. The test method is dragAndDropTarget. Start by loading the application: driver.get("https://jqueryui.com/droppable/");

There are 2 WebElements. First is WebElement source = driver.findElement(By.*id*("draggable")). The value for id is draggable. Next is the target. WebElement target = driver.findElement(By.*id*("droppable")); The value for id is droppable. Import WebElement. Did you notice that both WebElements were located inside an iFrame? Close the DOM and right click again. View Frame Source is an indicator that the WebElements are within a frame. Recall from the Switch To Frame Videos 29 – 32, we must switch to the frame before performing a command on the WebElement. If not, we will get an Exception while executing our Test Script.

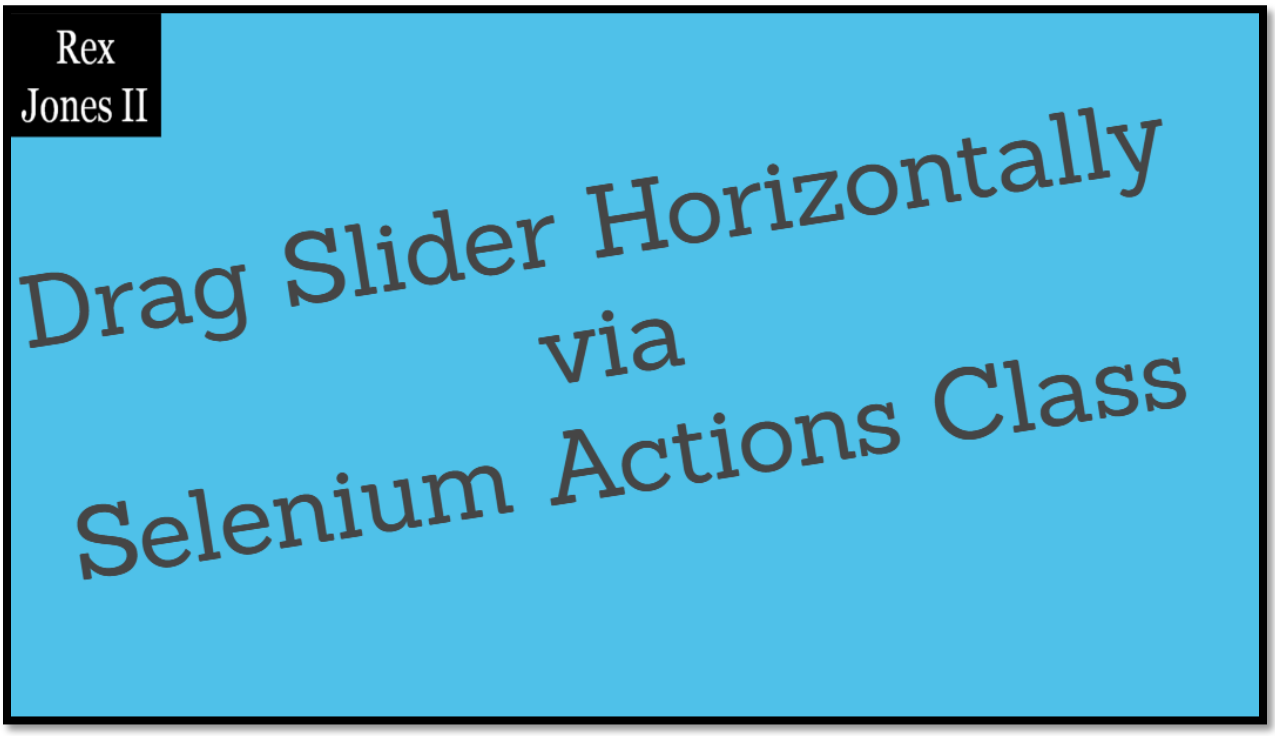
Let's write //iframe and we see there is only 1 iframe on this webpage. As a result, we can use the index to locate the only iFrame. Go back to Eclipse and add our switch statement. driver.switchTo().frame(0);

Now, we implement the Actions class with act as the object reference = new Actions driver in the parenthesis. Import the Actions class. Object Reference act dot dragAndDrop. Here we go, the drag and drop methods. Look at the description “A convenience method that performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.” The parameters are source and target. Next, we perform the action by writing perform.

Let’s Run. That’s It for Dragging Then Dropping A Target

Rex Jones II

Drag Slider Horizontally



In this video, we are going to automate dragging this slider to the left and right using Selenium's Actions class. Also get the value. The application is range slider.

Before we test, let's load the AUT by writing driver.get("https://rangeslider.js.org/");

Start by writing our WebElements. WebElement slider and WebElement output. Now let's find the element. Inspect the slider and write //div[@id='js-rangeslider-0']/div[2]. Bingo – Copy the value.

Go back to Eclipse. = driver.findElement(By.*xpath*("//div[@id='js-rangeslider-0']/div[2]"));
driver.findElement(By) Inspect the output and the value is js-output. driver.findElement(By.*id*("js-output"));

Next, we the add the Actions class with object reference act = new Actions (driver). act and our method is dragAndDropBy. Let's look at the syntax. In the previous session, we used dragAndDrop to move the source to the target. In this session, we are going to move the source using the xOffset. The description states "A convenience method that performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse." From the parameters section, we see x and y are the offsets. x moves horizontally which is left and right while the y moves vertically up and down.

The source is slider and lets moves the slider left by making x negative 100, and y is 0. Finally, we perform the action by selecting perform. Let's print the output so we can see the value in the console. sysout System.*out*.println("What Is The Output After Dragging Left? " + output.getText()); Let's Run. The Console shows 110 after dragging slider left.

I'm going to copy and paste the code for moving the slider left. Our method will drag the slider right. So I'm going to change left to right for the method name and print statement then change negative 100 to positive 100. Let's Run. Now the console shows 110 after dragging slider left and 490 after dragging slider right. Next, we will automate dragging a slider up and down.

Rex Jones II

Drag Slider Vertically

In this video, we are going to automate dragging this slider up and down using the Selenium's Action class. We will use the Range Sliders Foundation application.

Inspect the slider. It has a span tag with a class attribute. This time, I'm going to use CSS and not XPath. Write span.range-slider-handle and there are 11 results. Let's bypass the parent element and use the grandparent. div.small-3 space. I created another video called Ancestor-Child Relationship that explains this CSS value. Copy the value.

Go To Eclipse. Our first test will drag the slider down @Test / public void dragSliderDown (). WebElement slider = driver.findElement(By.*cssSelector*("div.small-3 span.range-slider-handle")); We are going to get the value after dragging the slider. So, let's inspect 50. The value for id is sliderOutput2. WebElement value = driver.findElement(By.*id*("sliderOutput2"));

Actions act = new Actions (driver); act.dragAndDropBy moves the source which is the slider using the xOffset or the yOffset. We are going to use the yOffset: (slider, 0, 34).perform(); Let's also print the value sysout("What Is The Value After Dragging The Slider Down? " + value.getText());

Rex
Jones II

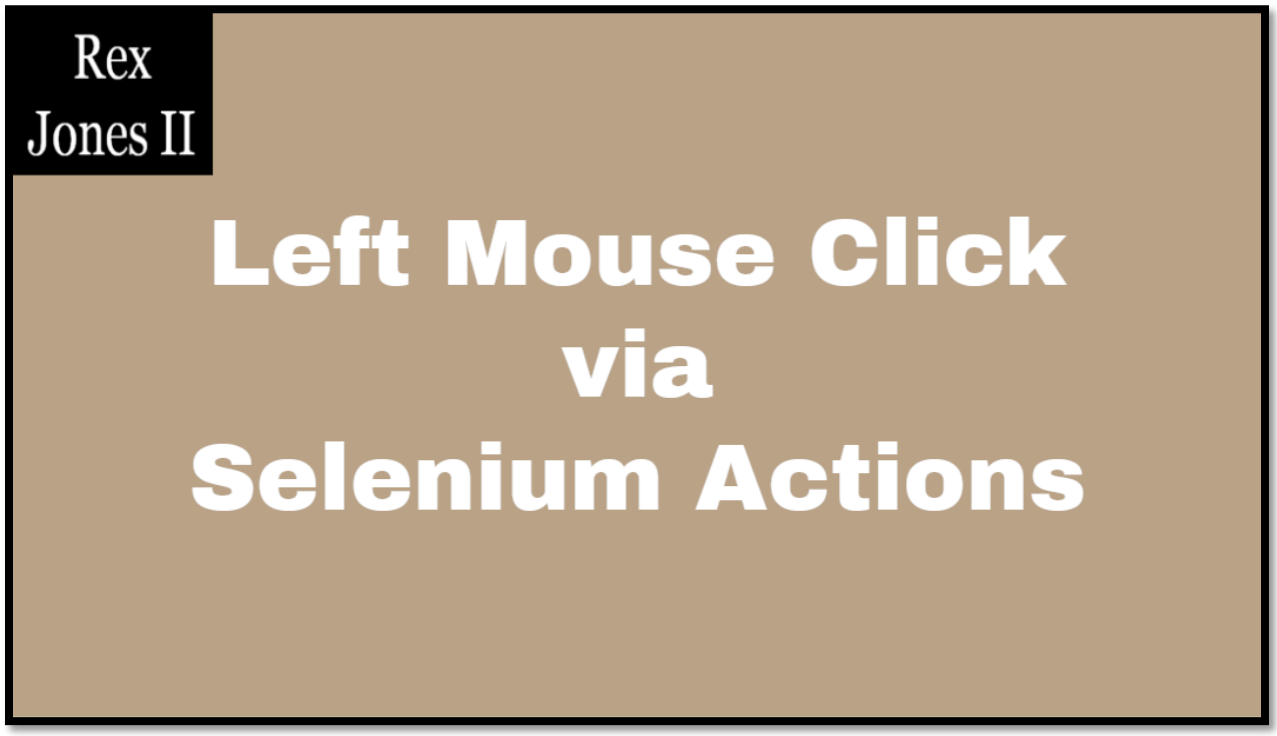
Load the AUT
driver.get("https://foundation.zurb.com/sites/docs/v/5.5.3/components/range_slider.html");

Now, Let's Run. We see the Console shows the new value 32.

Copy and Paste dragSliderDown. Change Down to Up and make 34 negative. Run again. The Console shows 32 for down and 70 for up. That's It and Thank You for watching How To Drag A Slider Vertically.

Rex
Jones II

Mouse Left Click



Rex Jones II

Introduction

Hello and Welcome. My name is Rex Jones II. Feel free to connect with me on LinkedIn and you can subscribe to my YouTube channel. In this video, we are going to automate clicking the left mouse button.

We see this (AUT) Application Under Test has a button called left click me. If I click the button, a list of items shows up. Our Test Script will left click the button and get the items. Let's go ahead and inspect the button. There's a span tag and a class attribute with a few values. We are going to use the last class name btn-neutral then write span.btn-neutral. Copy

Demo

Now, let's go to Eclipse. The setup method already has our AUT loaded
driver.get("https://swisnl.github.io/jQuery-contextMenu/demo/trigger-left-click.html");

Next, we find the WebElement by writing
WebElement button = driver.findElement(By.*cssSelector*("span.btn-neutral")); This is where the Actions class begins Actions and act as the object reference = new Actions (driver). Now, we left click the button. act.click and select the method that contains WebElement target. Our WebElement target is button then perform the action.

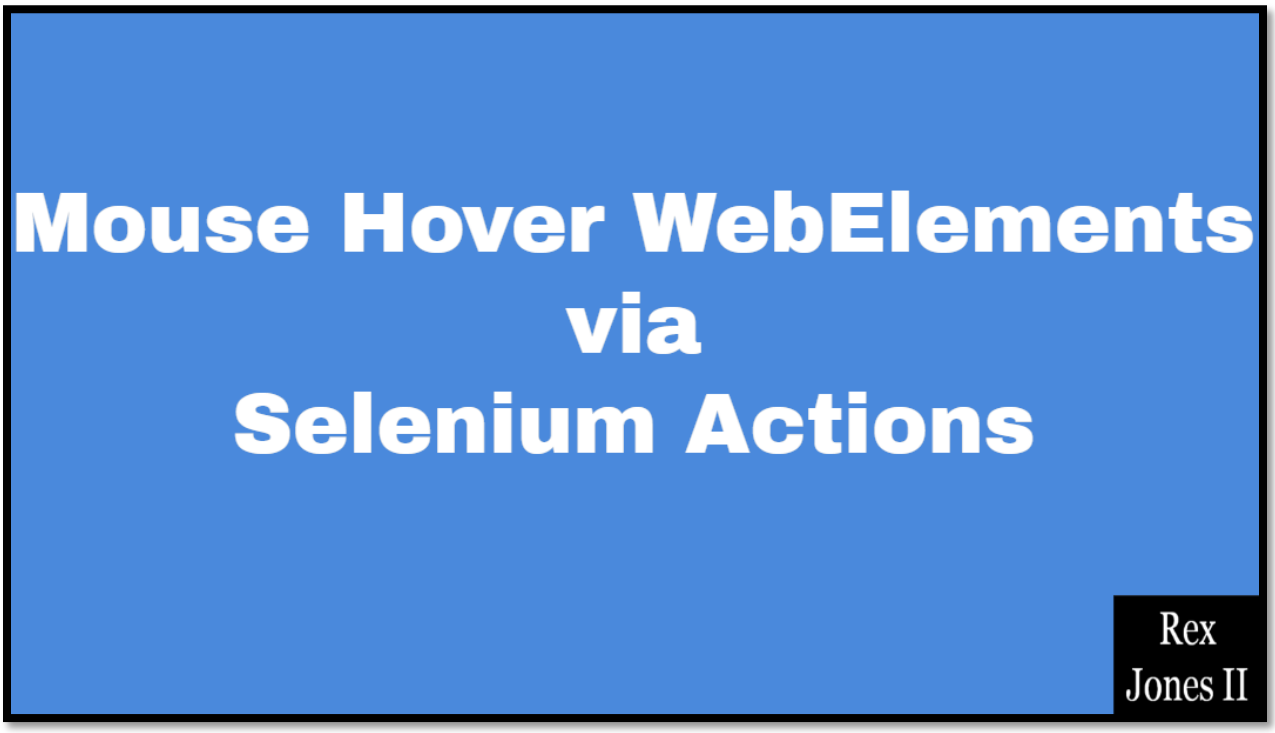
Next, we want to find the list of items after clicking the button. Let's move the Dock. Inspect the list. There's a span tag. Find the element by using CSS starting with the parent list item tag: li span and we see 6 results. The 6 results are for Edit, Cut, Copy, Paste, Delete, and Quit.

Now, go back to Eclipse and write our script.
driver.findElements(By.*cssSelector*("li span")) then assign the List of <WebElement> to elements =. I plan to create a video explaining the difference between findElement and findElements.

The last step is to print all of the items. We will use Java's Enhanced For Loop to iterate each element. for CTRL+SPACE change WebElement to element. I have a highlighter method in my utility package that will highlight each item. I'll make sure you have access to download the highlighter method. Highlighter.*highlightElement*(driver, element); then print each element sysout FSystem.*out*.println("\t" + element.getText()); Let's also print a note before starting the Enhanced For Loop. Sysout System.*out*.println("WebElements After Left Click:"); Now Run. We see each item after left clicking the button. That's it and Thank You for watching How To Use Selenium's Action Class to Left Click A Button.

Rex Jones II

Mouse Hovers WebElememts



Hello and Welcome. In this video, we are going to automate moving the mouse to hover over a WebElement. Selenium's Action class will help us move the mouse then perform an action. The Test Script will hover Hello, Sign In Account & List, select Your Account then get the page title. Inspect the element and we see nav-link-accountList as the id value. Copy the value.

Go to Eclipse and load the AUT driver.get("https://www.amazon.com/"); Next, we find the element
WebElement menuSignIn = driver.findElement(By.*id*("nav-link-accountList"));
Actions class act = new Actions (driver); Import the classes.
act dot then we move to the element by selecting moveToElement. Which element? menuSignIn then perform.

Go back to Amazon and Inspect Your Account. Let's find this value using XPath text function:
//span[text()='Your Account'] Bingo. Copy the value.

Go back to Eclipse. driver.findElement(By.*xpath*("//span[text()='Your Account']")).click(); The last statement will get and print the page title. sysout System.*out*.println("Title = " + driver.getTitle());

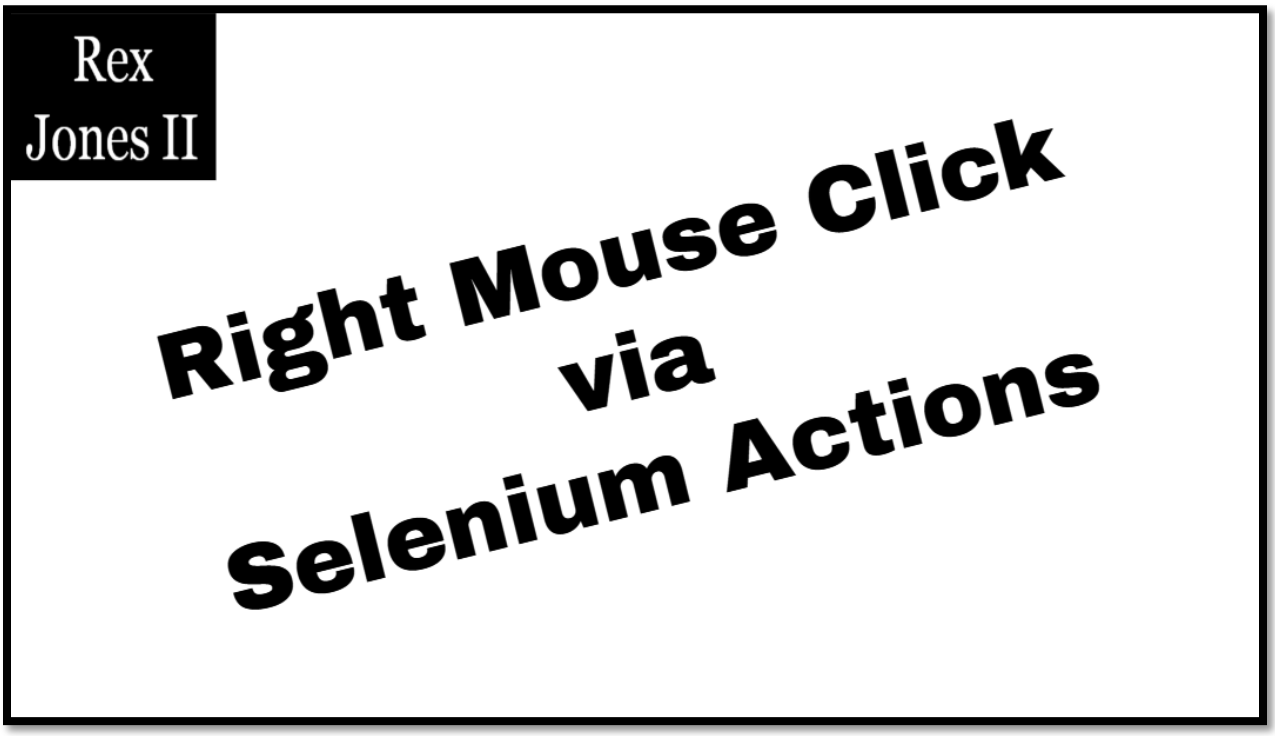
Rex Jones II

```
@Test
public void hoverAmazonMenu ()
{
   driver.get("https://www.amazon.com");

   WebElement menuSignIn = driver.findElement(By.id("nav-link-accountList"));

   Actions act = new Actions (driver);
   act.moveToElement(menuSignIn).perform();

   driver.findElement(By.xpath("//span[text()='Your Account']")).click();

   System.out.println("Title = " + driver.getTitle());
}
```

Let's Run. The Title is Your Account. That's it and Thank You for watching how to automate moving the mouse over an element.

Rex Jones II

Mouse
Right Click



Introduction

Hello and Welcome. My name is Rex Jones II. Connect with me on LinkedIn and you can subscribe to my YouTube channel. In this video, we are going to automate clicking the right mouse button using Selenium's Action class.

It's the same Test Script as the previous video left mouse click. We click the button and get the list of items. However, left click and right click use different methods. Inspect the button and we see the same span tag, class attribute, and class values as the left click. This time let's use a different value to find the element. How about class name btn? Copy the URL and

Demo

Go to Eclipse and load the application
driver.get("https://swisnl.github.io/jQuery-contextMenu/demo.html");

Find the button driver.findElement(By.*className*("btn")) and assign the WebElement to button =. Actions class with act as the object reference = new Actions (driver). Import Actions. Right click the

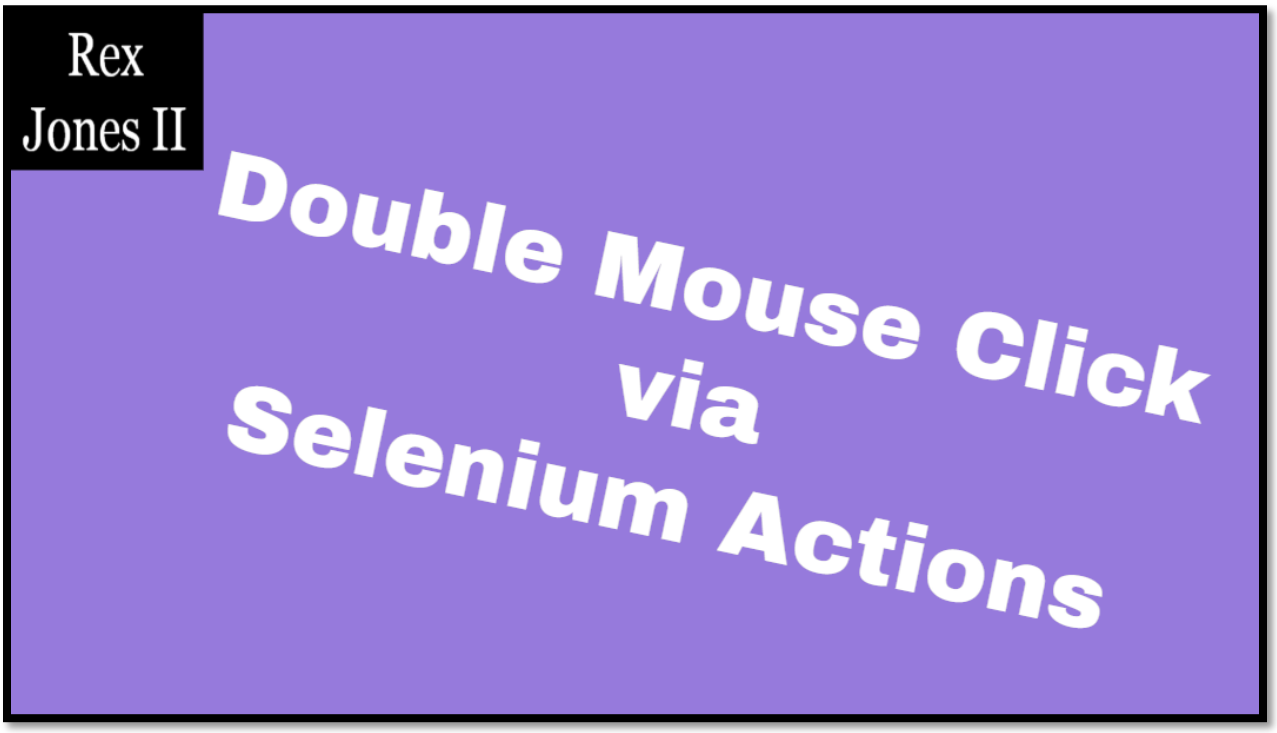
button by writing. act.contextClick. There are 2 methods but we will select WebElement target. The target is button.perform. Next is to find the list of items and print each item. I'm going to copy and paste from the Left Click class then change Left to Right.

```
@Test
public void rightClickButton ()
{
   WebElement button = driver.findElement(By.className("btn"));

   Actions act = new Actions (driver);
   act.contextClick(button).perform();

   List <WebElement> elements = driver.findElements(By.cssSelector("li span"));

   System.out.println("WebElements After Right Click:");

   for (WebElement element : elements)
   {
      Highlighter.highlightElement(driver, element);

      System.out.println("\t" + element.getText());
   }
}
```

Let's Run. All of the items Edit, Cut, Copy, Paste, Delete, and Quit showed up after Right Clicking the button. That's it and Thank You for watching How To Use Selenium's Action Class to Right Click A Button.

Rex Jones II

Mouse Double Click



Introduction

Hello and Welcome. My name is Rex Jones II. If you have not subscribed to my YouTube channel, click the Subscribe button then bell icon for each video notification. Also, connect with me on LinkedIn. In this video, we are going to automate double clicking a box using Selenium's Actions class.

jQuery is the AUT. I'm going to scroll to the bottom of this AUT and we see this blue box. The Test Script will get the color of the box, double click the box, then get the new color of the box. Inspect and we see View Frame Source shows us the box is located in an iframe. First, let's find the iframe by writing type 2 slashes, iframe. The application only has 1 frame. That makes it easy for us.

Demo

Go to Eclipse and the first thing we do is scroll to the bottom of the page using JavaScript Executor. Now, switch to the frame. driver.switchTo().frame It's best to switch to a frame using WebElement, name, or id but we only have 1 frame. So, let's use the index. (0); You can watch Videos 29 – 32 to learn more about switching to a frame.
After switching to the frame, we find the box but first let's import JavaScriptExecutor. Go back and inspect the box.

Within the iframe element, we can find the box using a direct relationship between the parent and child. body is the parent and div is the child. Therefore, we write body angle bracket div. There are 4 results but 3 of the results are outside the iframe. Let's search, search, search. 4 of 4 contains our element.

Let me show you how we get the color since we are inspecting the box. Select the Computed tab. Do you see how the background-color is yellow? We will use background-color as a parameter for the getCssValue Selenium method. The method will return the computed value which is 255, 255, 0.

Watch how the color changes after reloading the page and inspecting the box. It changed back to blue and we see rgb updated to 0, 0, 255.

Back to Eclipse. Find the box WebElement box = driver.findElement(By.*cssSelector*("body > div")); Now print the color.
Sysout System.*out*.println("Color Before: " + box.getCssValue We see the description shows Color values should be returned as rgba strings. RGBA stands for red green blue alpha. It's a 4-channel color model. The parameter - propertyName is background-color surrounded by double quotes.

Now, we double click by instantiating the Actions class act = new Actions (driver);
act.doubleClick and the WebElement target is box then we perform the action. The last step is to print color after double clicking the box. Copy and Paste then change Before to After.

```
@Test
public void doubleClickButton ()
{
   JavascriptExecutor js = (JavascriptExecutor) driver;
   js.executeScript("window.scrollBy(0, 2500)");

   driver.switchTo().frame(0);

   WebElement box = driver.findElement(By.cssSelector("body > div"));

   System.out.println("Color Before: " + box.getCssValue("background-color"));

   Actions act = new Actions (driver);
   act.doubleClick(box).perform();

   System.out.println("Color After: " + box.getCssValue("background-color"));
}
```

Let's Run. We see the colors are different before and after double clicking the box. That's it and Thank You for watching How To Double Clicking A WebElement Using Selenium Actions Class.