we will learn handling Keyboard and Mouse Event in Selenium Webdriver

**Action Class in Selenium**

**Action Class in Selenium** is a built-in feature provided by the selenium for handling keyboard and mouse events. It includes various operations such as multiple events clicking by control key, drag and drop events and many more. These operations from the action class are performed using the advanced user interaction API in Selenium Webdriver.

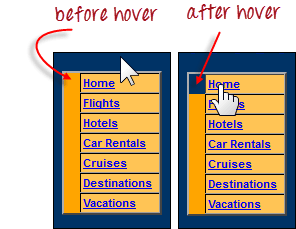
**Handling Keyboard & Mouse Events**

Handling special keyboard and mouse events are done using the **Advanced User Interactions API**. It contains the **Actions** and the **Action** classes that are needed when executing these events. The following are the most commonly used keyboard and mouse events provided by the Actions class.

|  |  |
| --- | --- |
| **Method** | **Description** |
| **clickAndHold()** | Clicks (without releasing) at the current mouse location. |
| **contextClick()** | Performs a context-click at the current mouse location. (Right Click Mouse Action) |
| **doubleClick()** | Performs a double-click at the current mouse location. |
| **dragAndDrop(source, target)** | Performs click-and-hold at the location of the source element, moves to the location of the target element, then releases the mouse.  **Parameters:**  source- element to emulate button down at.  target- element to move to and release the mouse at. |
| **dragAndDropBy(source, x-offset, y-offset)** | Performs click-and-hold at the location of the source element, moves by a given offset, then releases the mouse.  **Parameters**:  source- element to emulate button down at.  xOffset- horizontal move offset.  yOffset- vertical move offset. |
| **keyDown(modifier\_key)** | Performs a modifier key press. Does not release the modifier key – subsequent interactions may assume it’s kept pressed.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **keyUp(modifier \_key)** | Performs a key release.  **Parameters**:  modifier\_key – any of the modifier keys (Keys.ALT, Keys.SHIFT, or Keys.CONTROL) |
| **moveByOffset(x-offset, y-offset)** | Moves the mouse from its current position (or 0,0) by the given offset.  **Parameters**:  x-offset- horizontal offset. A negative value means moving the mouse left.  y-offset- vertical offset. A negative value means moving the mouse down. |
| **moveToElement(toElement)** | Moves the mouse to the middle of the element.  **Parameters**:  toElement- element to move to. |
| **release()** | Releases the depressed left mouse button at the current mouse location |
| **sendKeys(onElement, charsequence)** | Sends a series of keystrokes onto the element.  **Parameters**:  onElement – element that will receive the keystrokes, usually a text field  charsequence – any string value representing the sequence of keystrokes to be sent |

Learn Java Programming with Beginners Tutorial

In the following example, we shall use the moveToElement() method to mouse-over on one Mercury Tours’ table rows. See the example below.



The cell shown above is a portion of a <TR> element. If not hovered, its color is #FFC455 (orange). After hovering, the cell’s color becomes transparent. It becomes the same color as the blue background of the whole orange table.

**Step 1:**Import the **Actions** and **Action** classes.

Keyboard & Mouse Event using Action Class in Selenium Webdriver

**Step 2:**Instantiate a new Actions object.

Keyboard & Mouse Event using Action Class in Selenium Webdriver

**Step 3:**Instantiate an Action using the Actions object in step 2.

Keyboard & Mouse Event using Action Class in Selenium Webdriver

In this case, we are going to use the moveToElement() method because we are simply going to mouse-over the “Home” link. The build() method is always the final method used so that all the listed actions will be compiled into a single step.

**Step 4:**Use the perform() method when executing the Action object we designed in Step 3.

Keyboard & Mouse Event using Action Class in Selenium Webdriver

Below is the whole WebDriver code to check the background color of the <TR> element before and after the mouse-over.

package newproject;

import org.openqa.selenium.\*;

import org.openqa.selenium.firefox.FirefoxDriver;

import org.openqa.selenium.interactions.Action;

import org.openqa.selenium.interactions.Actions;

public class PG7 {

public static void main(String[] args) {

String baseUrl = "http://demo.guru99.com/test/newtours/";

System.setProperty("webdriver.gecko.driver","C:\\geckodriver.exe");

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement link\_Home = driver.findElement(By.linkText("Home"));

WebElement td\_Home = driver

.findElement(By

.xpath("//html/body/div"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr/td"

+ "/table/tbody/tr"));

Actions builder = new Actions(driver);

Action mouseOverHome = builder

.moveToElement(link\_Home)

.build();

String bgColor = td\_Home.getCssValue("background-color");

System.out.println("Before hover: " + bgColor);

mouseOverHome.perform();

bgColor = td\_Home.getCssValue("background-color");

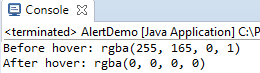
System.out.println("After hover: " + bgColor);

driver.close();

}

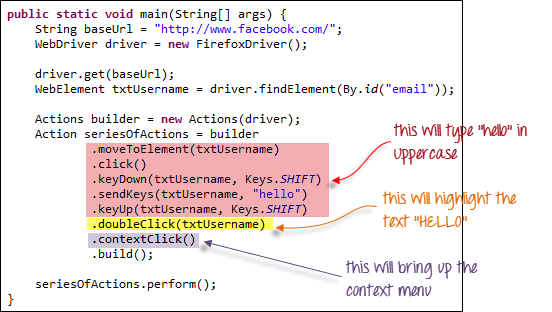
}

The output below clearly states that the background color became transparent after the mouse-over.



**Building a Series of Multiple Actions**

**You can build a series of actions using the Action and Actions classes**. Just remember to close the series with the build() method. Consider the sample code below.



public static void main(String[] args) {

String baseUrl = "http://www.facebook.com/";

WebDriver driver = new FirefoxDriver();

driver.get(baseUrl);

WebElement txtUsername = driver.findElement(By.id("email"));

Actions builder = new Actions(driver);

Action seriesOfActions = builder

.moveToElement(txtUsername)

.click()

.keyDown(txtUsername, Keys.SHIFT)

.sendKeys(txtUsername, "hello")

.keyUp(txtUsername, Keys.SHIFT)

.doubleClick(txtUsername)

.contextClick()

.build();

seriesOfActions.perform() ;

}

