**Handling Advanced Keyboard and Mouse events using Selenium:**

Selenium uses Webdriver commands like sendKeys (), click () to interact with the UI elements of the Web application. These commands will not be helpful in some user actions using Keyboard and mouse events. Some of the complex interactions like drag and drop, right clicking the web element and other actions using control and shift keys cannot be handled pertinently using selenium commands. To assist these user actions, Selenium uses the ***Actions*** class which is a built in feature of Selenium Web Driver.

**Actions Class:**

Actions class will give a direct interaction with the web element in order to access the complex user gestures. The keyboard and mouse events are handled using Advanced User Interaction API of Actions class. Some AUI are Mouse hover, double click etc.. which we can see in most of the web applications.

***What is an Actions class?***

**Actions class** is a class which implements the builder pattern to perform composite sequence of actions using mouse and keyboard. Actions class is an API which uses low level interface (Action) to give input in the web browsers. Interface used here is Action interface which denotes the single action performed by the user. To automate these keyboards and mouse action, first we need to instantiate the Actions class as below;

**Actions act=new Actions (driver) ;** webdriver must be passed as argument

Actions class will be imported from the selenium package

**org.openqa.selenium.interactions.Actions;**

**Methods in Actions Class:**

**Perform() –** Perform method must be defined at the end of all the user actions so that actual interaction with the element is done by keyboard and mouse events.

***Methods which enables mouse events***

[**Click()**](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_Interactions_Actions_Click.htm) **–click the current co-ordinates of the mouse pointer**

[**Click(WebElement)**](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_Interactions_Actions_Click_1.htm)- **clicks the specified web Element.**

[**clickAndHold()**](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_Interactions_Actions_ClickAndHold.htm) **–Click and hold mouse button at the specified pointer co-ordinates.**

[**ClickAndHold(IWebElement**)](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_Interactions_Actions_ClickAndHold_1.htm) – **Click and hold the mouse button at the identified web Element.**

**ContextClick() – to enable the right click**

**contextClick(WebElement) – to enable the right click at the identified web Element.**

**doubleClick() – helps to double click the element.**

**doubleClick(WebElement target) – to double click at the middle of the identified web element.**

[**dragAndDrop**](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_Interactions_Actions_DragAndDrop.htm)**(WebElement source, WebElement target) – to click and hold the element(moves) from one location to other location.**

[**dragAndDrop**](https://www.selenium.dev/selenium/docs/api/dotnet/html/M_OpenQA_Selenium_Interactions_Actions_DragAndDrop.htm)**by(WebElement source, int x offset, int y offset) – to click and hold the element(moves) from one position to other position and then releases the button**

**x offset- shows the horizontal moves**

**y offset-shows the vertical moves**

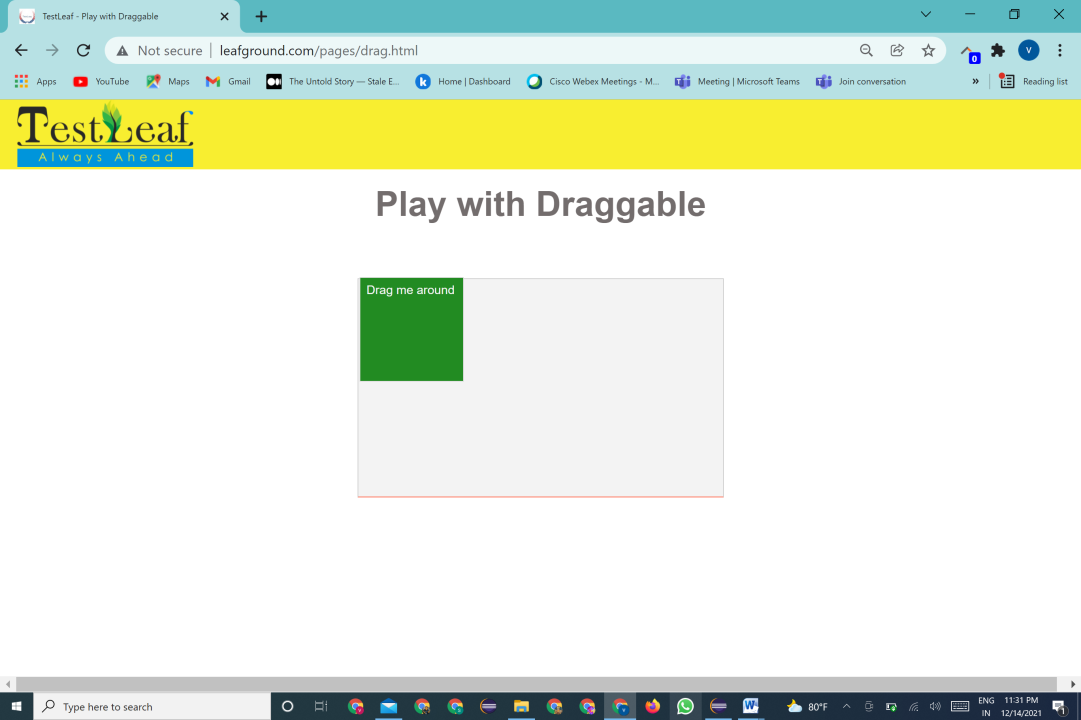
**moveByOffset(int x offset, int y offset) –moves the mouse from the current position to given offset.**

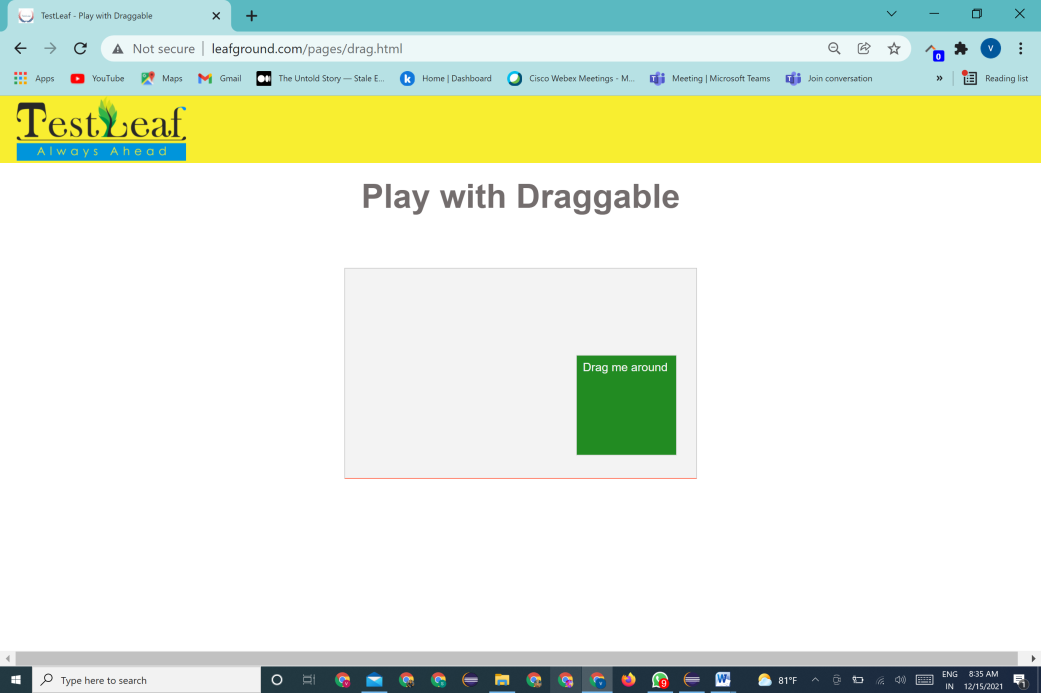
**x offset- shows the horizontal moves(negative value show the move towards left)**

**y offset-shows the vertical moves(negative value show the move upwards)**

**moveToElement(WebElement target) - to assist in mouse hovering the element and get into the Element view.**

**moveToElement(WebElement target, int x offset, int y offset)-** **Moves the mouse to an offset from the element's in-view center point.**

**For instance, lets the example**

****

In the above fig the element **Drag me around** has to be moved to the desired location within in the given space. The action involved here is to click and hold the element and then move to required location.To automate this action we need to do the basic configuaration and then instantiate the action class.

import java.util.concurrent.TimeUnit;

import org.openqa.selenium.By;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import org.openqa.selenium.interactions.Actions;

import io.github.bonigarcia.wdm.WebDriverManager;

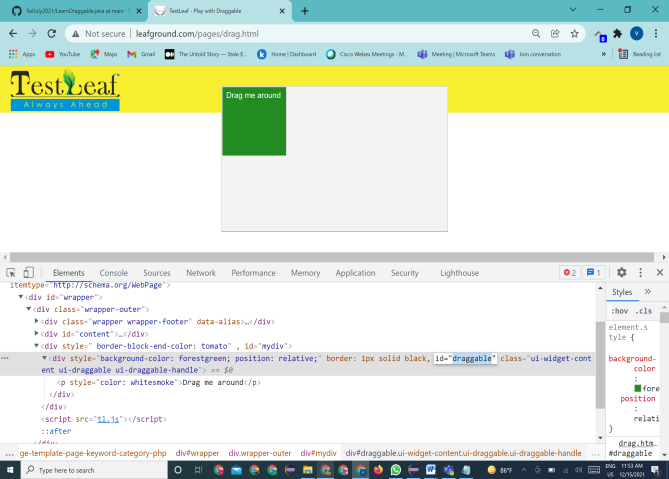
public class LearnDraggable {

public static void main(String[] args) {

WebDriverManager.chromedriver().setup();

ChromeDriver driver = **new** ChromeDriver();

driver.get("httpwww.leafground.compagesdrag.html");

driver.manage().window().maximize();

driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

//instantiate the actions class

Actions builder = new Actions(driver);

WebElement drag = driver.findElement(By.id(draggable));

builder.dragAndDropBy(drag, 100, 100).perform();

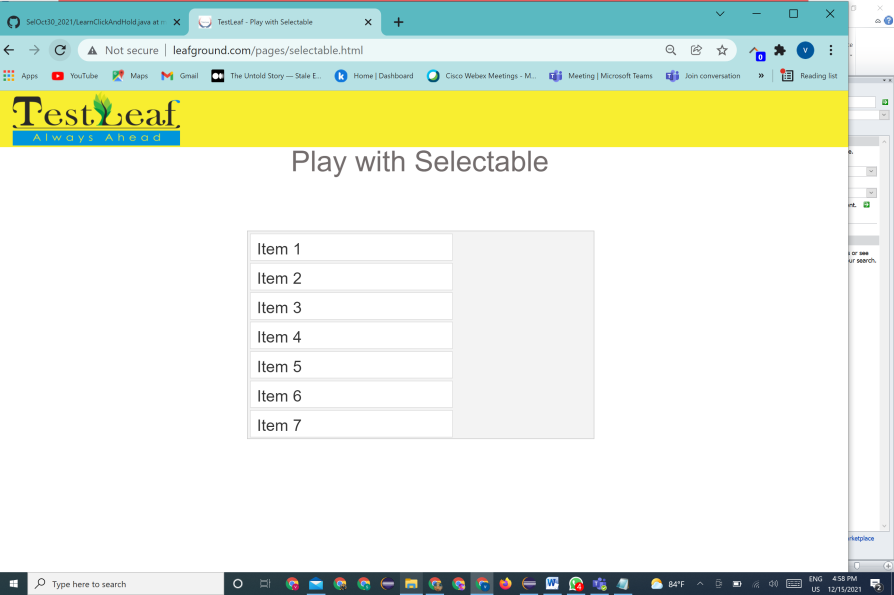
**x and y offset value**

release(): Releases the depressed left mouse button at the existing mouse location

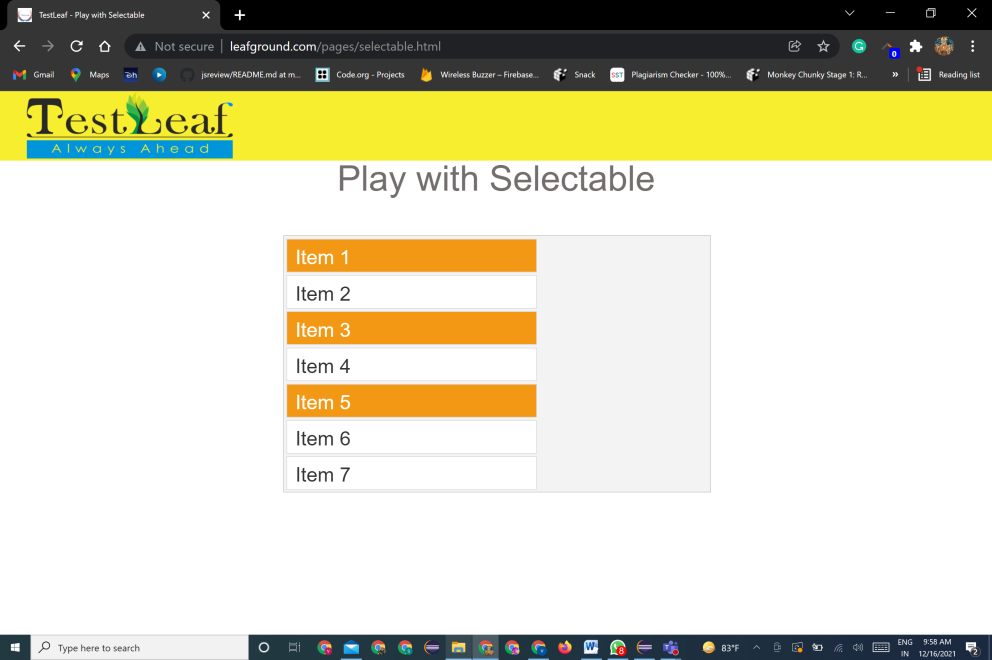
**Some of the Keyboard events are,**

**keyDown(charSequence key)** –press and hold the required key in the keyboard without releasing it. Subsequent actions may assume it as pressed**. Some key actions involves Keys.shift,Keys.Cntrl,keys.Alt**

**sendKeys**(**keysToSend**) - sends a series of keystrokes onto the element

**keyUp(charSequence key)** –performs the key release action.****

**In the below application, if we need to select the item 1 item 2 and item 3, our action key will be control+click ..to automate this action, inspect the required webElement pass it as argument to the action sequence as below**



WebElement item1 = driver.findElement(By.xpath("//li[text()='Item 1']"));

WebElement item3 = driver.findElement(By.xpath("//li[text()='Item 3']"));

WebElement item5 = driver.findElement(By.xpath("//li[text()='Item 5']"));

//Create object for Actions class and pass the driver as argument

Actions builder = new Actions(driver);

builder .keyDown(Keys.CONTROL) to press the control key

.click(item1)

.click(item3)

.click(item5) to release the contol key

.keyUp(Keys.CONTROL).perform();