Practical No. 9 Action Class in Selenium Webdriver

Date: ________________

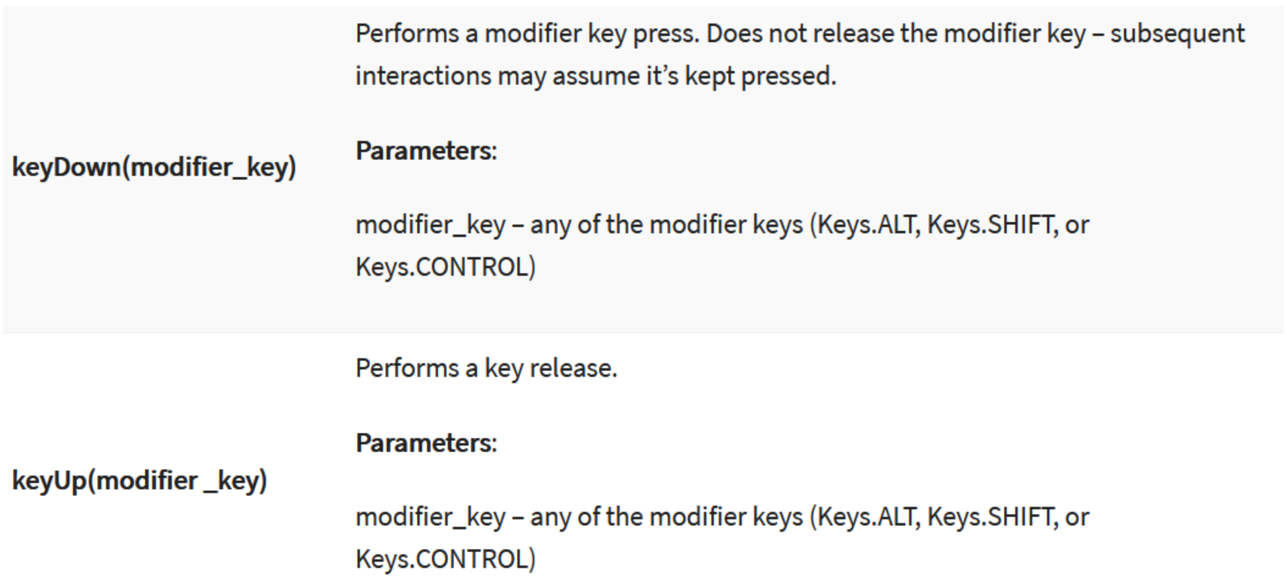
Aim:

To learn Action Class in Selenium, a built-in feature provided by the selenium for handling keyboard and mouse events.

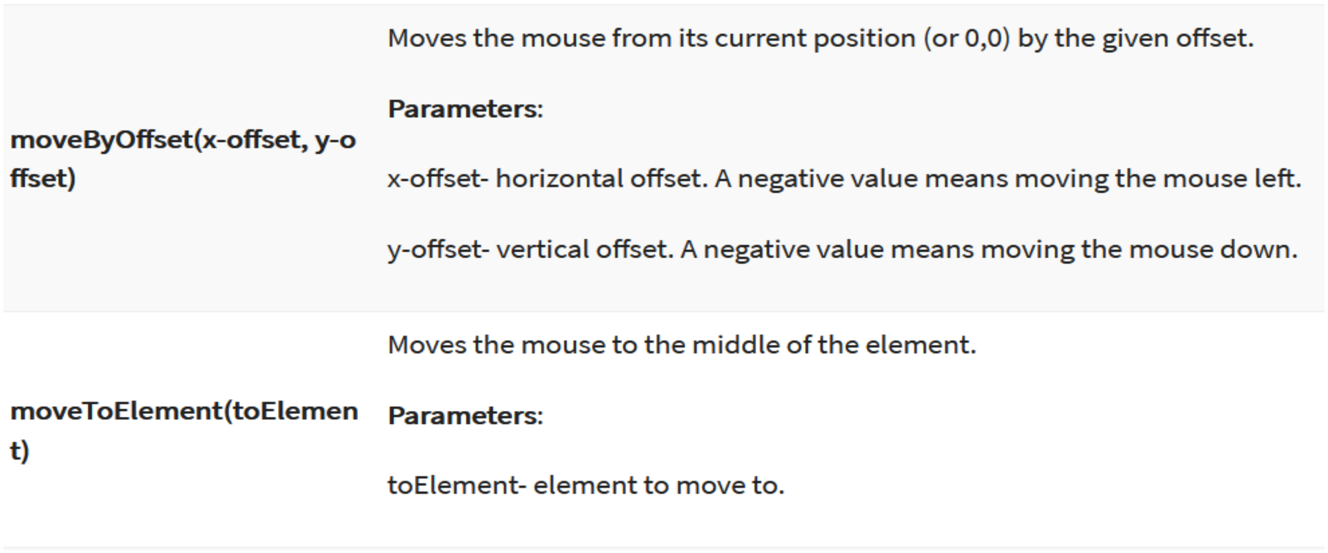
Theory:

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 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.

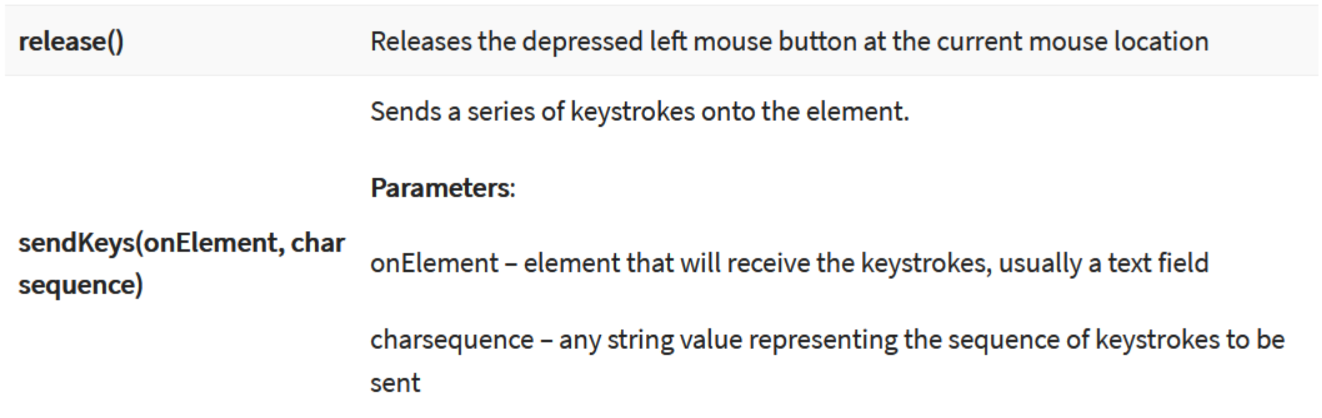
Method	**Description**
clickAndHold()	Clicks (without releasing) at the current mouse location.
contextClick()	Performs a context-click at the current mouse location. (Right Click Mouse Action)
doubleClick()	Performs a double-click at the current mouse location.
dragAndDrop(source, target)	Performs click-and-hold 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, char sequence)	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

Steps to use Actions Classes

Step 1: Import the Actions and Action classes.

Step 2: Instantiate a new Actions object.

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

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

Following Actions are available in Actions class

The moveByOffset action

The moveByOffset() method is used to move the mouse from its current position to another point on the web page.

Developers can specify the X distance and Y distance the mouse has to be moved.

When the page is loaded, generally the initial position of a mouse would be (0, 0), unless there is an explicit focus declared by the page.

The API syntax for the moveByOffset() method is as follows:

```
public Actions moveByOffset(int xOffSet, int yOffSet)
```

In the preceding code, xOffSet is the input parameter providing the WebDriver the amount of offset to be moved along the x axis. A positive value is used to move the cursor to the right, and a negative value is used to move the cursor to the left.

yOffSet is the input parameter providing the WebDriver the amount of offset to be moved along the y axis. A positive value is used to move the cursor down along the y axis and a negative value is used to move the cursor toward the top.

When the xOffSet and yOffSet values result in moving the cursor out of the document, a MoveTargetOutOfBoundsException is raised

The click at current location action

The click() method is used to simulate the left-click of your mouse at its current point of location.

This method doesn't really realize where or on which element it is clicking. It just blindly clicks wherever it is at that point of time.

Hence, this method is used in combination with some other action rather than independently, to create a composite action.

The API syntax for the click() method is as follows:

```
public Actions click()
```

The click() method doesn't really have any context about where it is performing its action; hence, it doesn't take any input parameter.

The click on a WebElement action

When the WebElement has its own identifiers, such as a name or ID, we can use another overloaded version of the click() method to click directly on the WebElement.

The API syntax for clicking on a WebElement is as follows:

public Actions click(WebElement onElement)

The input parameter for this method is an instance of the WebElement on which the click action should be performed.

This method, like all the other methods in the Actions class, will return an Actions instance.

The clickAndHold at current location action

The clickAndHold() method is another method of the Actions class that left-clicks on an element and holds it without releasing the left button of the mouse.

This method will be useful when executing operations such as drag-and-drop.

This method is one of the variants of the clickAndHold() method that the Actions class provides.

The clickAndHold a WebElement action

WebDriver provides the developers with another variant or overloaded method of the clickAndHold() method that takes the WebElement as input.

The API syntax is as follows:

public Actions clickAndHold(WebElement onElement)

The input parameter for this method is the WebElement that has to be clicked and held.

The return type, as in all the other methods of the Actions class, is the Actions instance.

The release at current location action

The ultimate action that has to be taken on a held WebElement is to release it so that the element can be dropped or released from the mouse.

The release() method is the one that can release the left mouse button on a WebElement.

The API syntax for the release() method is as follows:

public Actions release()

The preceding method doesn't take any input parameter and returns the Actions class instance.

The release on another WebElement action

This is an overloaded version of the release() method. Using this, you can actually release the currently held WebElement in the middle of another WebElement.

In this way, we don't have to calculate the offset of the target WebElement from the held WebElement.

The API syntax is as follows:

public Actions release(WebElement onElement)

The input parameter for the preceding method is obviously the target WebElement where the held WebElement should be dropped.

The return type is the instance of the Actions class.

The moveToElement action

The moveToElement() method is another method of WebDriver that helps us to move the mouse cursor to a WebElement on the web page.

The API syntax for the moveToElement() method is as follows:

public Actions moveToElement(WebElement toElement)

The input parameter for the preceding method is the target WebElement where the mouse should be moved.

The dragAndDropBy action

WebDriver has given us a convenient out of the box method to use. Let's see its API syntax.

The API syntax for the dragAndDropBy() method is as follows:

public Actions dragAndDropBy(WebElement source,int xOffset,int yOffset)

The WebElement input parameter is the target WebElement to be dragged, the xOffset parameter is the horizontal offset to be moved, and the yOffset parameter is the vertical offset to be moved.

The dragAndDrop action

The dragAndDrop() method is similar to the dragAndDropBy() method.

The only difference being that instead of moving the WebElement by an offset, we move it on to a target element.

The API syntax for the dragAndDrop() method is as follows:

public Actions dragAndDrop(WebElement source, WebElement target)

The input parameters for the preceding method are the WebElement source and the WebElement target, while the return type is the Actions class.

The doubleClick at current location action

doubleClick() is another out of the box method that WebDriver provides to emulate the double-clicking of the mouse.

The API syntax is as follows:

public Actions doubleClick()

This method doesn't take any input parameters, as it just clicks on the current cursor location and returns an Actions class instance.

The doubleClick on WebElement action

Now that we have seen a method that double-clicks at the current location, we will discuss another method that WebDriver provides to emulate the double-clicking of a WebElement.

The API syntax for the doubleClick() method is as follows:

public Actions doubleClick(WebElement onElement)

The input parameter for the preceding method is the target WebElement that has to be double-clicked and the return type is the Actions class

The contextClick on WebElement action

The contextClick() method, also known as right-click, is quite common on many web pages these days. The context is nothing but a menu; a list of items is associated to a WebElement based on the current state of the web page.

This context menu can be accessed by a right-click of the mouse on the WebElement.

WebDriver provides the developer with an option of emulating that action using the contextClick() method.

The API syntax for the contextClick() method is as follows:

public Actions contextClick(WebElement onElement)

The input parameter is obviously the WebElement that has to be right-clicked, and the return type is the Actions instance.

The contextClick at current location action

The API syntax for the contextClick() method is as follows:

public Actions contextClick()

As expected, the preceding method doesn't expect any input parameter, and returns the Actions instance.

Keyboard-based interactions KeyDown , KeyUp , SendKeys

The keyDown and keyUp actions

The keyDown() method is used to simulate the action of pressing and holding a key.

The keys that we are referencing here are Shift, Ctrl, and Alt keys.

The keyUp() method is used to release the key that is already pressed using the keyDown() method.

The API syntax for the keyDown() method is as follows

public Actions keyDown(Keys theKey) throws IllegalArgumentException

An IllegalArgumentException is thrown when the passed key is not one of the Shift, Ctrl, and Alt keys.

The API syntax for the keyUp() method is as follows

public Actions keyUp(Keys theKey)

The keyUp action performed on a key, on which a keyDown action is not already being performed, will result in some unexpected results.

So, we must make sure we perform the keyUp action after a keyDown action is performed

Implementation

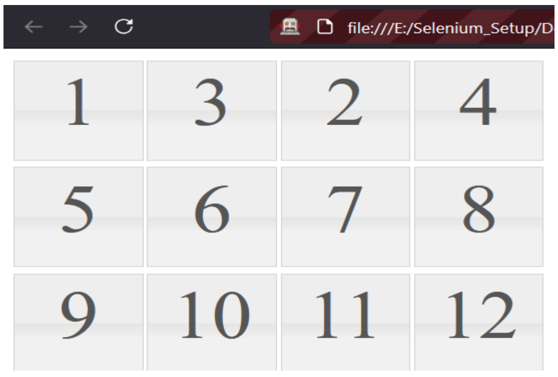
1. **Write a selenium script to move tile3 to the position of tile2 on Sortable.html**

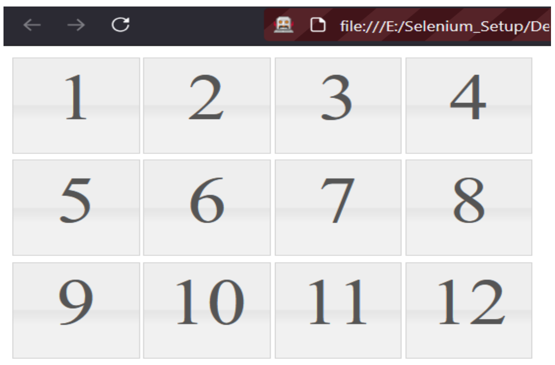
Code :

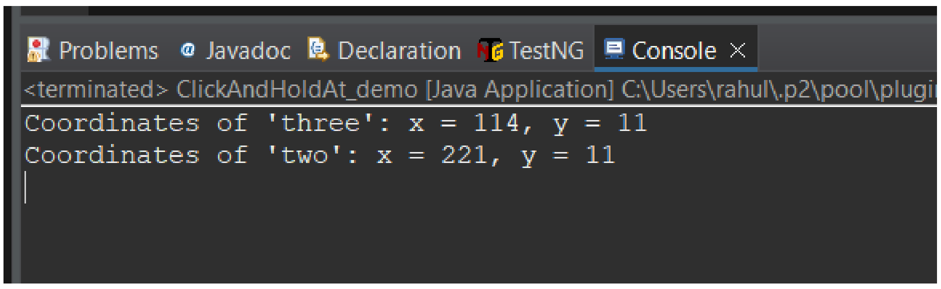
```
package selenium_test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;
import org.openqa.selenium.Point;

public class ClickAndHoldAt_demo {

  public static void main(String[] args) throws InterruptedException {
    // Set path of Gecko driver
    System.setProperty("webdriver.gecko.driver", "E:\\Selenium_setup\\geckodriver.exe");
```

```
        // Create an instance of the Firefox driver
        WebDriver driver = new FirefoxDriver();

        driver.get("file:///E:/Selenium_Setup/Demo%20pages/Sortable.html");

        WebElement three = driver.findElement(By.name("three"));
        Thread.sleep(2000);
        WebElement two = driver.findElement(By.name("two"));

        // Get the x and y coordinates of the elements
        Point threeLocation = three.getLocation();
        Point twoLocation = two.getLocation();

        System.out.println("Coordinates of 'three': x = " + threeLocation.getX() + ", y = " +
    threeLocation.getY());
        System.out.println("Coordinates of 'two': x = " + twoLocation.getX() + ", y = " +
    twoLocation.getY());
        Actions builder = new Actions(driver);
        // Perform the click and hold action
        builder.clickAndHold(two).moveToElement(three).release().perform();
        // Close the driver        driver.quit(); } }
```

Output :







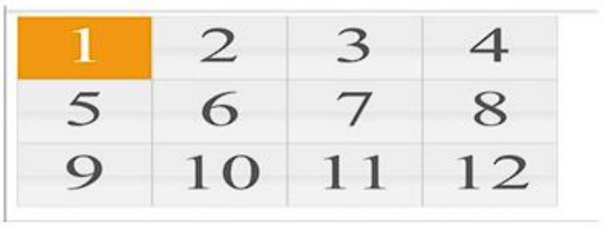
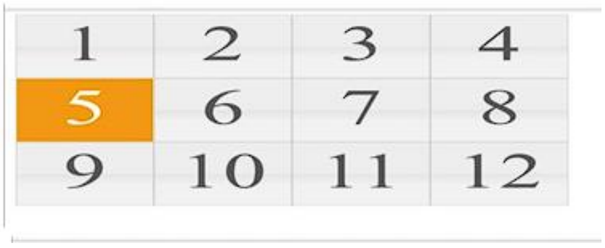
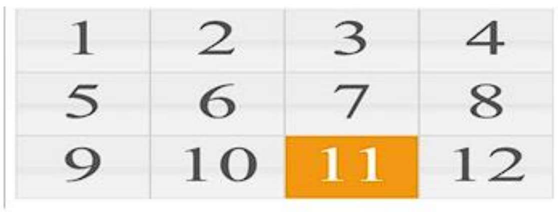
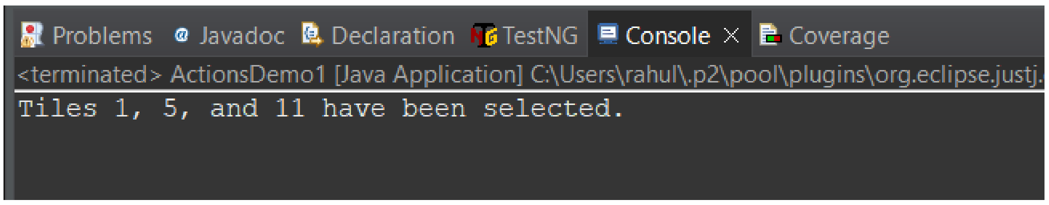
```
<terminated> ClickAndHoldAt_demo [Java Application] C:\Users\rahul\.p2\pool\plugi
Coordinates of 'three': x = 114, y = 11
Coordinates of 'two': x = 221, y = 11
```

2. Write a selenium script to select tile 1, tile 5, tile 11 on Selectable.html

Code :

```
package selenium_test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class ActionsDemo1 {
  public static void main(String[] args) throws InterruptedException {
    // Set path for Chrome driver
    System.setProperty("webdriver.gecko.driver", "E:\\Selenium_setup\\geckodriver.exe");

    // Create an instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();

    // Navigate to the HTML page
    driver.get("file:///E://Selenium_Setup//Demo pages//Selectable.html");
              //driver.get("file:///C:/Users/rahul/Downloads/HTML/HTML/Selectable.html");

    // Wait for a moment to ensure the page is fully loaded
    Thread.sleep(2000);


    // Create an Actions object
    Actions actions = new Actions(driver);


    // Locate the tiles using their names
    WebElement tile1 = driver.findElement(By.name("one"));    // Tile 1
    WebElement tile5 = driver.findElement(By.name("five"));   // Tile 5
    WebElement tile11 = driver.findElement(By.name("eleven")); // Tile 11
    // Select tiles 1, 5, and 11
    actions.click(tile1)
        .click(tile5)
        .click(tile11)
        .perform(); // Execute the action

    // Wait for a moment to observe the selection
    Thread.sleep(3000);

    // Optional: Print confirmation
    System.out.println("Tiles 1, 5, and 11 have been selected.");

    // Close the driver
    driver.quit(); } }
```

Output :

```
<terminated> ActionsDemo1 [Java Application] C:\Users\rahul\.p2\pool\plugins\org.eclipse.justj.
Tiles 1, 5, and 11 have been selected.
```

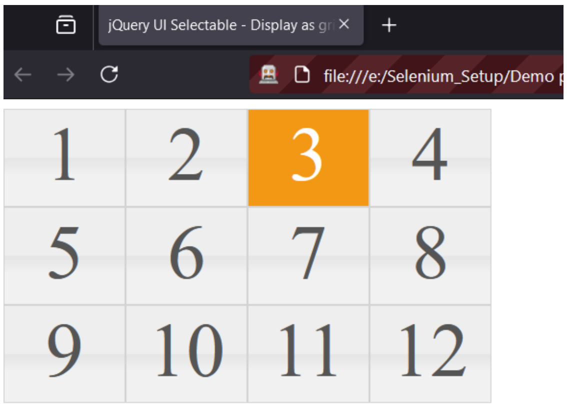
3. Write a selenium script to select tile 3 using moveByOffset() method and select tile 11 using click on web Element method click() on Selectable.html

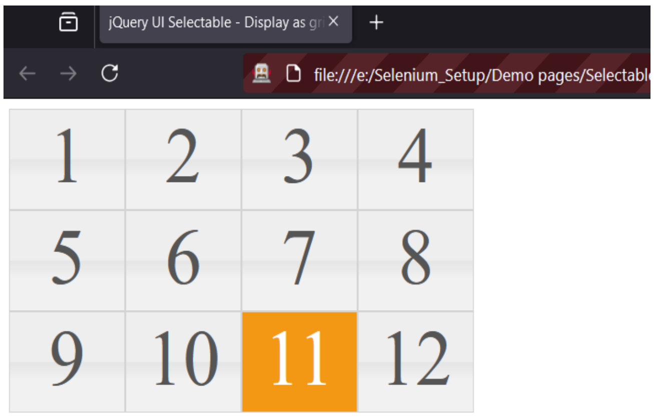
Code :

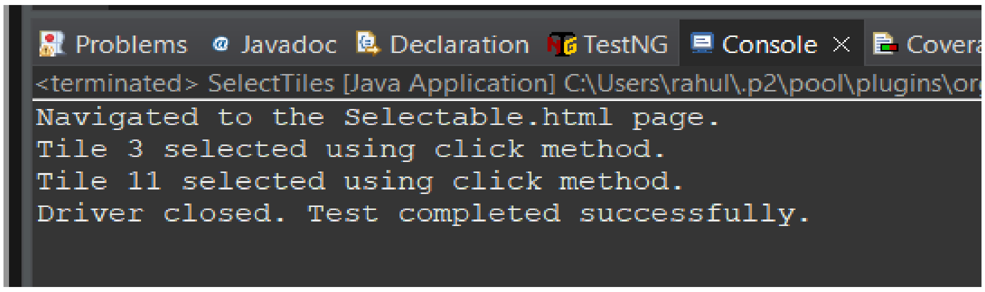
```
package selenium_test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class SelectTiles {
  public static void main(String[] args) throws InterruptedException {
    // Set path for GeckoDriver
    System.setProperty("webdriver.gecko.driver", "E:\\Selenium_setup\\geckodriver.exe");

    // Create an instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();

    // Navigate to the specified URL
    driver.get("E:\\Selenium_Setup\\Demo pages\\Selectable.html");
    System.out.println("Navigated to the Selectable.html page.");

    // Create an Actions object
    Actions actions = new Actions(driver);

    // Select tile 3 directly
    WebElement tile3 = driver.findElement(By.xpath("//li[text()='3']"));
    tile3.click();
    System.out.println("Tile 3 selected using click method.");
    Thread.sleep(2000);

    // Locate tile 11 and click on it using click() method
    WebElement tile11 = driver.findElement(By.xpath("//li[text()='11']"));
```

```
        tile11.click();
        System.out.println("Tile 11 selected using click method.");
        Thread.sleep(2000);

        // Close the driver
        driver.quit();
        System.out.println("Driver closed. Test completed successfully."); } }
```

Output :


jQuery UI Selectable - Display as gr
file:///e:/Selenium_Setup/Demo p
1
2
3
4
5
6
7
8
9
10
11
12


jQuery UI Selectable - Display as gr
file:///e:/Selenium_Setup/Demo pages/Selectabl
1
2
3
4
5
6
7
8
9
10
11
12



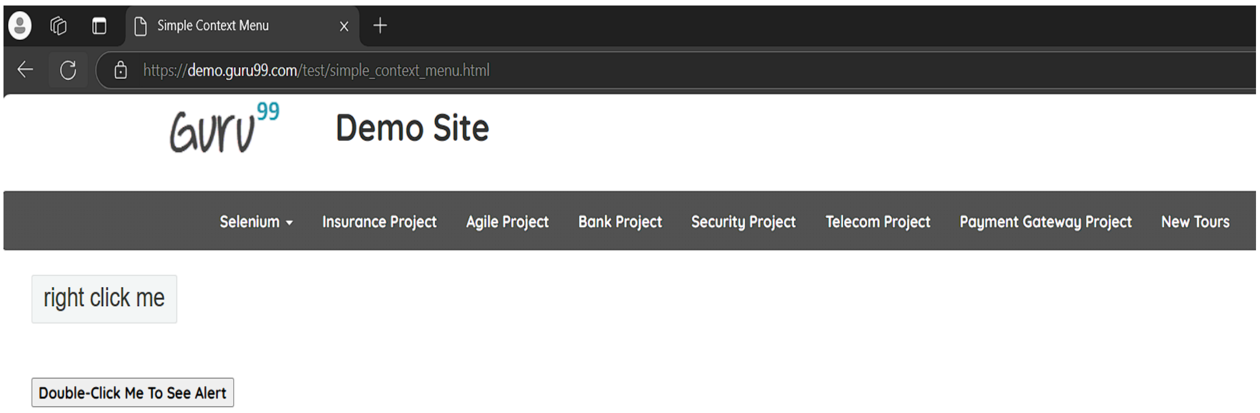
```
<terminated> SelectTiles [Java Application] C:\Users\rahul\.p2\pool\plugins\or
Navigated to the Selectable.html page.
Tile 3 selected using click method.
Tile 11 selected using click method.
Driver closed. Test completed successfully.
```

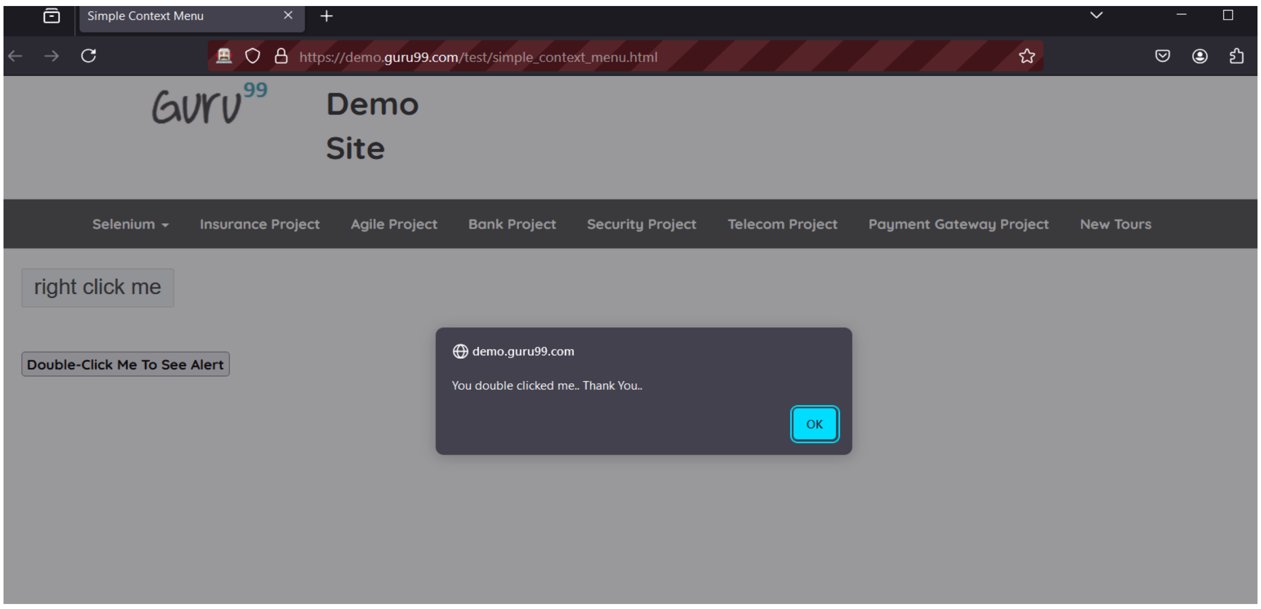
4. **Write a selenium script to double click and right click the elements on http://demo.guru99.com/test/simple_context_menu.html**

Code :

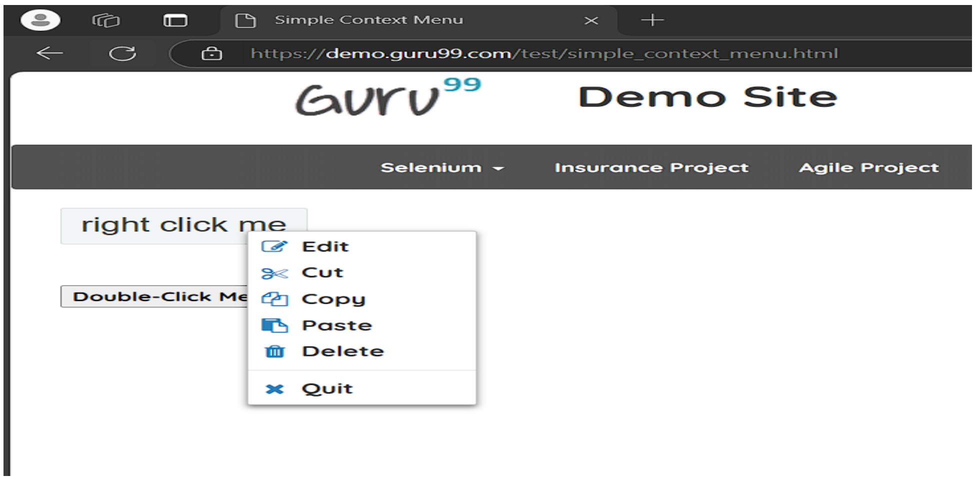
```
package selenium_test;
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class ContextMenuDemo {
  public static void main(String[] args) throws InterruptedException {
        // Set path for Chrome driver
    System.setProperty("webdriver.gecko.driver", "E:\\Selenium_setup\\geckodriver.exe");

    // Create an instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();

    // Navigate to the specified URL
    driver.get("http://demo.guru99.com/test/simple_context_menu.html");

    // Locate the element to be double-clicked
    WebElement doubleClickElement = driver.findElement(By.xpath("//button[text()='Double-
Click Me To See Alert']"));
    // Create an Actions object
    Actions actions = new Actions(driver);
    // Double click on the element
    actions.doubleClick(doubleClickElement).perform();
    // Handle the alert
    Alert alert = driver.switchTo().alert();
    System.out.println("Alert Text: " + alert.getText());
    Thread.sleep(4000);
    alert.accept();
    // Locate the element to be right-clicked
    WebElement rightClickElement = driver.findElement(By.xpath("//span[text()='right click
me']"));
    // Right click on the element
    actions.contextClick(rightClickElement).perform();

    // Close the driver
    driver.quit();  } }
```

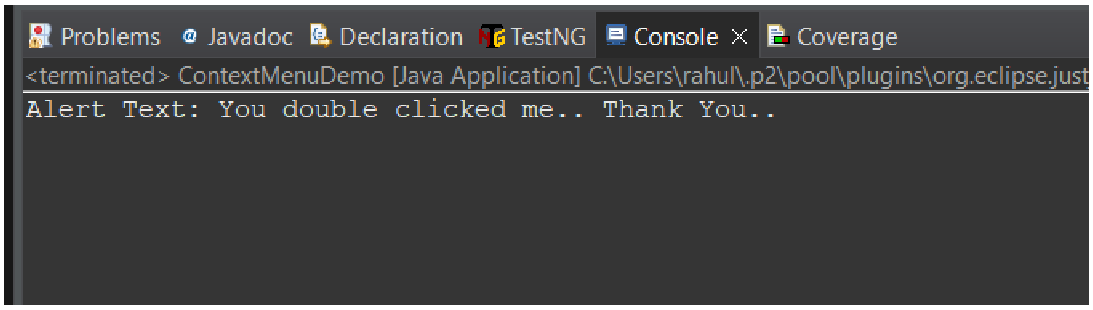
Output :





You double clicked me.. Thank You..





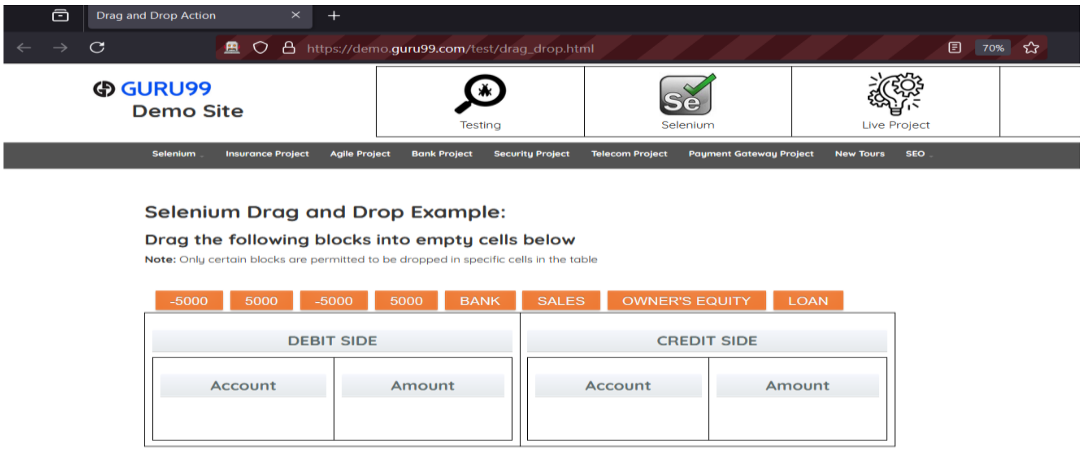
```
<terminated> ContextMenuDemo [Java Application] C:\Users\rahul\.p2\pool\plugins\org.eclipse.just
Alert Text: You double clicked me.. Thank You..
```

5. **Write a selenium script to drag the BANK element and drop on the DEBIT SIDE block through dragAndDrop method on the following webpage http://demo.guru99.com/test/drag_drop.html. Use xpath to locate required elements.**

Code :

```
package selenium_test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class Bank {
  public static void main(String[] args) throws InterruptedException {
    // Set path for Firefox driver
    System.setProperty("webdriver.gecko.driver", "E:\\Selenium_setup\\geckodriver.exe");

    // Create an instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();

    // Navigate to the webpage
    System.out.println("Navigating to the webpage...");
    driver.get("http://demo.guru99.com/test/drag_drop.html");
    Thread.sleep(3000);

    // Create an instance of Actions class
    Actions actions = new Actions(driver);

    // Locate the BANK element and DEBIT SIDE block
    System.out.println("Locating the BANK element...");
    WebElement bankElement = driver.findElement(By.xpath("//a[text()=' BANK ']"));
    Thread.sleep(3000);
    System.out.println("BANK element located.");

    System.out.println("Locating the DEBIT SIDE block...");
    WebElement debitSideBlock = driver.findElement(By.xpath("//ol[@id='bank']"));
    Thread.sleep(3000);
    System.out.println("DEBIT SIDE block located.");


    // Perform the drag and drop action
    System.out.println("Performing drag and drop action...");
    actions.dragAndDrop(bankElement, debitSideBlock).perform();
    System.out.println("Drag and drop action performed.");
    // Wait to see the result
    Thread.sleep(3000);

    // Close the browser
    System.out.println("Closing the browser...");      driver.quit();
    System.out.println("Browser closed."); } }
```

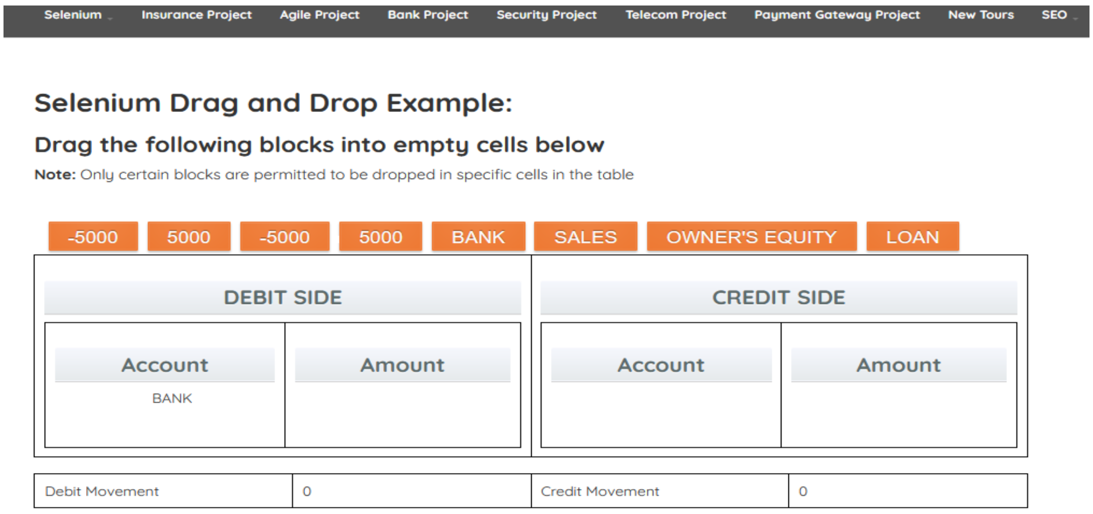
Output :



Selenium Drag and Drop Example:

Drag the following blocks into empty cells below

Note: Only certain blocks are permitted to be dropped in specific cells in the table

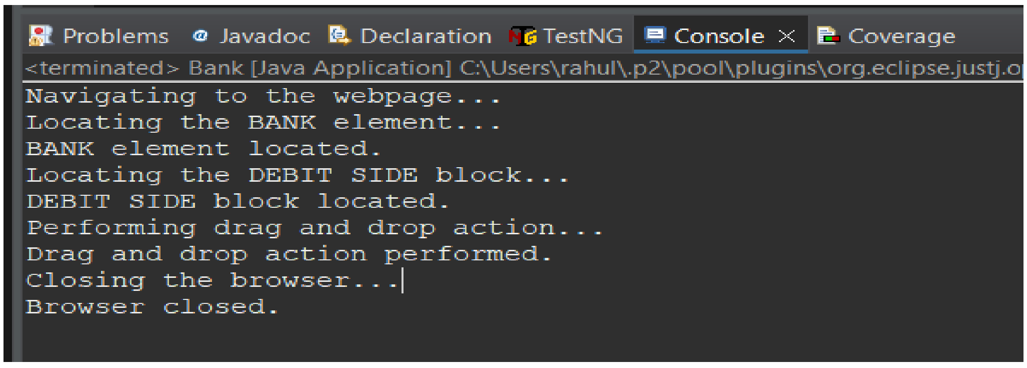


Selenium Drag and Drop Example:

Drag the following blocks into empty cells below

Note: Only certain blocks are permitted to be dropped in specific cells in the table

Debit Movement	0	Credit Movement	0



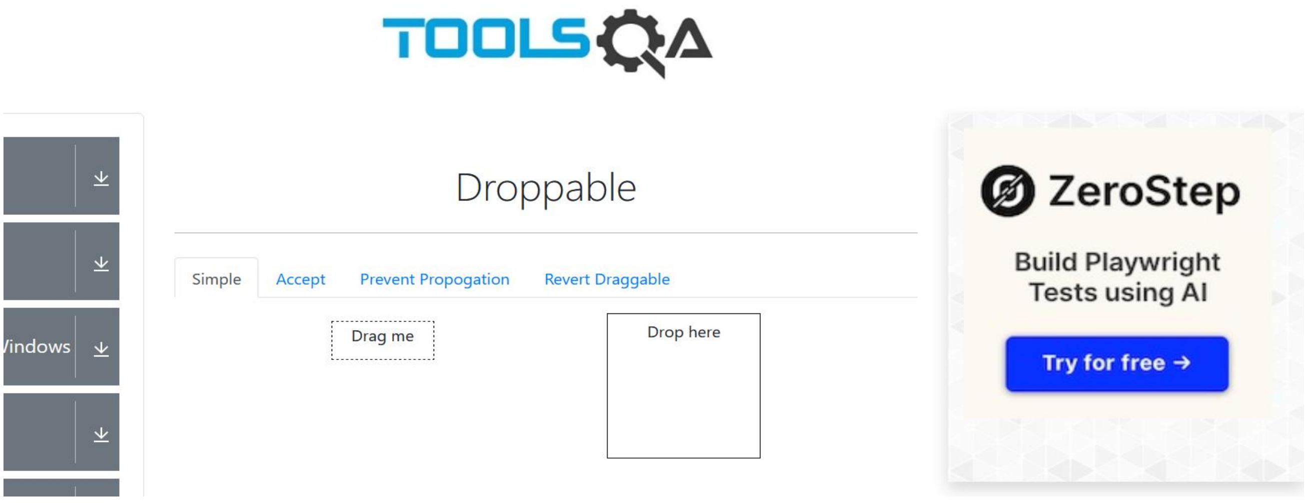
```
<terminated> Bank [Java Application] C:\Users\rahul\.p2\pool\plugins\org.eclipse.justj.o
Navigating to the webpage...
Locating the BANK element...
BANK element located.
Locating the DEBIT SIDE block...
DEBIT SIDE block located.
Performing drag and drop action...
Drag and drop action performed.
Closing the browser...
Browser closed.
```

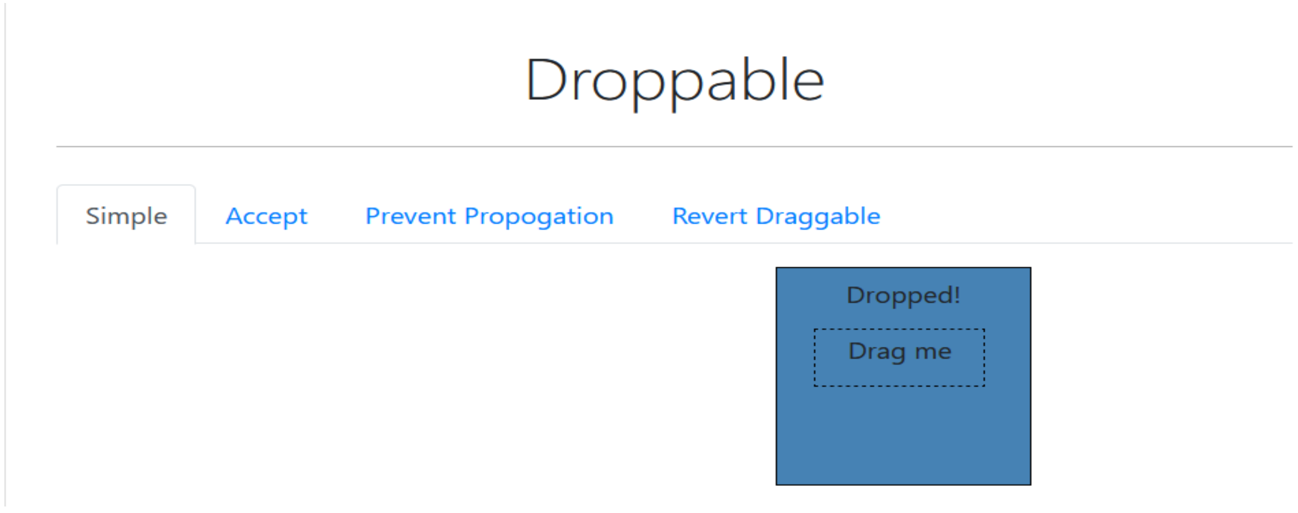
6. Write a selenium script to implement drag and drop action on https://demoqa.com/droppable/

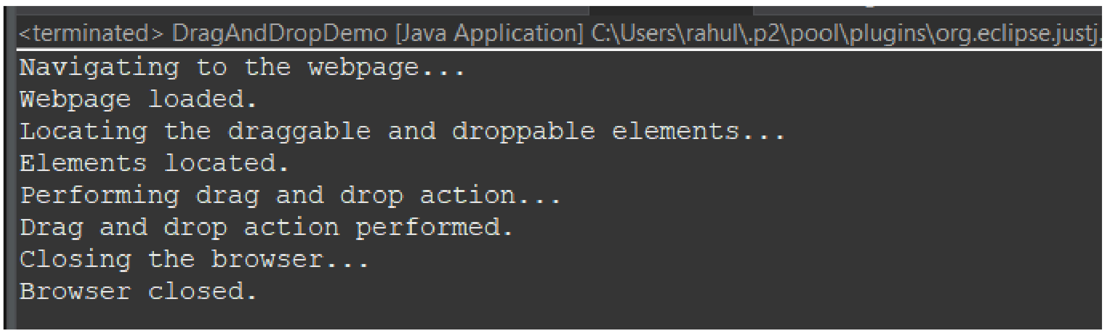
Code:

```
package selenium_test;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.interactions.Actions;

public class DragAndDropDemo {
  public static void main(String[] args) throws InterruptedException {
    // Set path for Firefox driver
    System.setProperty("webdriver.gecko.driver", "E:\\Selenium_setup\\geckodriver.exe");

    // Create an instance of the Firefox driver
    WebDriver driver = new FirefoxDriver();

    // Navigate to the webpage
    System.out.println("Navigating to the webpage...");
    driver.get("https://demoqa.com/droppable/");
    Thread.sleep(3000);
    System.out.println("Webpage loaded.");

    // Create an instance of Actions class
    Actions actions = new Actions(driver);

    // Locate the draggable and droppable elements
    System.out.println("Locating the draggable and droppable elements...");
    WebElement draggable = driver.findElement(By.id("draggable"));
    WebElement droppable = driver.findElement(By.id("droppable"));
    Thread.sleep(3000);
    System.out.println("Elements located.");
    // Perform the drag and drop action
    System.out.println("Performing drag and drop action...");
    actions.dragAndDrop(draggable, droppable).perform();
    System.out.println("Drag and drop action performed.");

    // Wait to see the result
    Thread.sleep(3000);

    // Close the browser
    System.out.println("Closing the browser...");      driver.quit();
    System.out.println("Browser closed."); } }
```

Output :







```
<terminated> DragAndDropDemo [Java Application] C:\Users\rahul\.p2\pool\plugins\org.eclipse.justj.
Navigating to the webpage...
Webpage loaded.
Locating the draggable and droppable elements...
Elements located.
Performing drag and drop action...
Drag and drop action performed.
Closing the browser...
Browser closed.
```

Conclusion: Learnt advance mouse and keyboard interactions using Selenium.

After performing this Practical/lab, students are expected to answer following questions

Q.1 What is Actions class in Selenium?

Q.2 How to build a set of actions in a composite action to execute it in one pass?