**Actions Class:**

**org.openqa.selenium.interactions.Actions** or simply **Actions** class supports user to perform all the operations related with mouse and keyboard keys from the keysboard.   
Actions class extends Object class directly and TouchActions class extends Actions class. It follows multilevel inheritance. Actions class will not help in window switch and frome switching operations  
  
**Implements the builder pattern:** Builds a CompositeAction containing all actions specified by the method calls.  
**Composite Actions :** User can create series of operations using Actions class and force them work one by one continuously using a actions builder (build() method from the Actions class helps the user to perform the pattern building).  
  
**Explanation :** The Actions class allows us to build a chain of actions that we would like ti perform.  
This means that we can build up a nice sequence of operations, for example, "Press Shift key and type something and then release", or if we want to work checkboxes, we can press CTRL and then click the required checkboxes  
  
**How to use Actions class in Selenium webdriver**

1. Create object to Actions class to access all the non-static methods, Actions class must be imported from **import org.openqa.selenium.interactions.Actions;**

**Actions act = new Actions(driver);**

Operations Supported by Actions Class

**Mouse Actions:**  
Mouse Actions on Selenium webdriver helps user to perform all the operations related with mouse  
Mouse Operations like : clicking, dragging, moving, clicking and dragging, hovering, double clicking, right clicking.

1.build()  
2. click()  
3. clickAndHold()  
4. contextClick()  
5. doubleClick()  
6. dragAndDrop(WebElement source, WebElement target)  
7. moveByOffset(int xOffset, int yOffset)  
8. moveToElement(WebElement target)  
9. moveToElement(WebElement target, int xOffset, int yOffset)  
10. pause(long pause)  
11. perform()  
12. release()

|  |  |
| --- | --- |
| **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 |

**KeyBoard Actions**

Selenium Webdriver Provides Api for keyboard as well through action class, with help of it we can emulate keyboard but here selenium webdriver limits the keys to only **CTRL, ALT, Shift** with keyUp, KeyDown methods  
  
But with help of Sendkeys method we can still automate all the keys present in the keyboard,

1. sendKeys(java.lang.CharSequence... keys)  
   2. keyUp(java.lang.CharSequence key)  
   3. keyDown(java.lang.CharSequence key)

*Complete code to context click the link*

**public class MouseRightClick {**

**public static void main(String[] args) {**

**// set the geckodriver.exe property**

**System.setProperty("webdriver.gecko.driver", "C:/~/geckodriver.exe");**

**// open firefox**

**WebDriver driver = new FirefoxDriver();**

**driver.manage().timeouts().implicitlyWait(1, TimeUnit.MINUTES);**

**// navigate to url**

**driver.get("URL");**

**//find the element to context click**

**WebElement element = driver.findElement(By.linkText("object"));**

**// Create object for Actions class**

**Actions act = new Actions(driver);**

**// call the contextClick method and pass the stored parameter**

**act.contextClick(element);**

**// comlete the operation**

**act.perform();**

**}**

**}**

If you try to click an element which is not displayed on the screen, selenium webdriver throws org.openqa.selenium.interactions.MoveTargetOutOfBoundsException

*Complete code to****Drag and Drop****click the link*

**// PLEASE DO WRITE IMPORT STATEMENTS**

**public class MouseDragAndDrop {**

**public static void main(String[] args) {**

**// set chrome driver exe path**

**System.setProperty("webdriver.chrome.driver", "C:/~/chromedriver.exe");**

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

**driver.manage().timeouts().implicitlyWait(1, TimeUnit.MINUTES);**

**// navigate to url**

**driver.get("URL");**

**//find the source element**

**WebElement source = driver.findElement(By.xpath(“obj1”)**

**WebElement target = driver.findElement(By.id("obj2"));**

**// Create object for Actions class**

**Actions act = new Actions(driver);**

**// perform drag and drop**

**act.dragAndDrop(source, target).perform();**

**}**

**}**