**Methods Summary**

Summary of methods for Actions class, is listed with syntax, explanation, parameters and return value

**keyDown**

public Actions **keyDown(java.lang.CharSequence key)**

Performs a modifier key press. Does not release the modifier key – subsequent interactions may assume it’s kept pressed. Note that the modifier key is **never released implicitly** – either **keyUp(theKey) or sendKeys(Keys.NULL) must be called to release the modifier**.

Parameters:

key – Either Keys.SHIFT, Keys.ALT or Keys.CONTROL. If the provided key is none of those, **IllegalArgumentException** is thrown.

Returns: A self-reference.

keyDown

public Actions **keyDown(WebElement target,   java.lang.CharSequence key)**

Performs a modifier key press after focusing on an element. Equivalent to: Actions.click(element).sendKeys(theKey);

Parameters:

key – Either Keys.SHIFT, Keys.ALT or Keys.CONTROL. If the provided key is none of those, IllegalArgumentException is thrown.

target – WebElement to perform the action

Returns: A self-reference.

keyUp

public Actions **keyUp(java.lang.CharSequence key)**

Performs a modifier key release. Releasing a non-depressed modifier key will yield undefined behaviour.

Parameters:

key – Either Keys.SHIFT, Keys.ALT or Keys.CONTROL.

Returns: A self-reference.

keyUp

public Actions **keyUp(WebElement target, java.lang.CharSequence key)**

Performs a modifier key release after focusing on an element. Equivalent to: Actions.click(element).sendKeys(theKey);

Parameters:

key – Either Keys.SHIFT, Keys.ALT or Keys.CONTROL.

target – WebElement to perform the action on

Returns: A self-reference.

sendKeys

public Actions **sendKeys(java.lang.CharSequence… keys)**

Sends keys to the active element. This differs from calling WebElement.sendKeys(CharSequence…) on the active element in two ways:

The modifier keys included in this call are not released.

There is no attempt to re-focus the element – so sendKeys(Keys.TAB) for switching elements should work.

Parameters:

keys – The keys.

Returns: A self-reference.

Throws: **java.lang.IllegalArgumentException –** if keys is null

sendKeys

public Actions **sendKeys(WebElement target,java.lang.CharSequence… keys)**

Equivalent to calling: Actions.click(element).sendKeys(keysToSend). This method is different from WebElement.sendKeys(CharSequence…) – see sendKeys(CharSequence…) for details how.

Parameters:

target – element to focus on.

keys – The keys.

Returns: A self-reference.

Throws: java.lang.IllegalArgumentException – if keys is null

clickAndHold

public Actions **clickAndHold(WebElement target)**

Clicks (without releasing) in the middle of the given element. This is equivalent to: Actions.moveToElement(onElement).clickAndHold()

Parameters:

target – Element to move to and click.

Returns: A self-reference.

clickAndHold

public Actions **clickAndHold()**

Clicks (without releasing) at the current mouse location.

Returns: A self-reference.

release

public Actions **release(WebElement target)**

Releases the depressed left mouse button, in the middle of the given element. This is equivalent to: Actions.moveToElement(onElement).release() Invoking this action without invoking clickAndHold() first will result in undefined behaviour.

Parameters:

target – Element to release the mouse button above.

Returns: A self-reference.

release

public Actions **release()**

Releases the depressed left mouse button at the current mouse location.

Returns: A self-reference.

click

public Actions **click(WebElement target)**

Clicks in the middle of the given element. Equivalent to: Actions.moveToElement(onElement).click()

Parameters:

target – Element to click.

Returns: A self-reference.

click

public Actions **click()**

Clicks at the current mouse location. Useful when combined with moveToElement(org.openqa.selenium.WebElement, int, int) or moveByOffset(int, int).

Returns: A self-reference.

doubleClick

public Actions **doubleClick(WebElement target)**

Performs a double-click at middle of the given element. Equivalent to: Actions.moveToElement(element).doubleClick()

Parameters: target – Element to move to.

Returns: A self-reference.

doubleClick

public Actions **doubleClick()**

Performs a double-click at the current mouse location.

Returns: A self-reference.

moveToElement

public Actions **moveToElement(WebElement target)**

Moves the mouse to the middle of the element. The element is scrolled into view and its location is calculated using getBoundingClientRect.

Parameters: target – element to move to.

Returns: A self-reference.

moveToElement

public Actions **moveToElement(WebElement target, int xOffset, int yOffset)**

Moves the mouse to an offset from the top-left corner of the element. The element is scrolled into view and its location is calculated using getBoundingClientRect.

Parameters:

target – element to move to.

xOffset – Offset from the top-left corner. A negative value means coordinates left from the element.

yOffset – Offset from the top-left corner. A negative value means coordinates above the element.

Returns: A self-reference.

moveByOffset

public Actions **moveByOffset(int xOffset,int yOffset)**

Moves the mouse from its current position (or 0,0) by the given offset. If the coordinates provided are outside the viewport (the mouse will end up outside the browser window) then the viewport is scrolled to match.

Parameters:

xOffset – horizontal offset. A negative value means moving the mouse left.

yOffset – vertical offset. A negative value means moving the mouse up.

Returns: A self-reference.

Throws: MoveTargetOutOfBoundsException – if the provided offset is outside the document’s boundaries.

contextClick

public Actions **contextClick(WebElement target)**

Performs a context-click at middle of the given element. First performs a mouseMove to the location of the element.

Parameters:

target – Element to move to.

Returns: A self-reference.

contextClick

public Actions **contextClick()**

Performs a context-click at the current mouse location.

Returns: A self-reference.

dragAndDrop

public Actions **dragAndDrop(WebElement source, WebElement target)**

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.

Parameters:

source – element to emulate button down at.

target – element to move to and release the mouse at.

Returns: A self-reference.

dragAndDropBy

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

A convenience method that 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.

Returns:A self-reference.

pause

public Actions **pause(long pause)**

Performs a pause.

Parameters:pause – pause duration, in milliseconds.

Returns:A self-reference.

pause

public Actions pause(java.time.Duration duration)

tick

public Actions tick(Interaction… actions)

tick

public Actions tick(Action action)

build

public Action build()

Generates a composite action containing all actions so far, ready to be performed (and resets the internal builder state, so subsequent calls to build() will contain fresh sequences).

Returns:the composite action

perform

public void perform()

A convenience method for performing the actions without calling build() first.

Steps to using Actions

* Import Actions and Action class from the Advanced User Interactions API as

import org.openqa.selenium.interactions.Actions;

import org.openqa.selenium.interactions.Action;

* Actions object instantiation. Call the Actions class constructor that accepts current driver object as input parameter

Actions builder = new Actions(driver);

* Use of Action object to perform mouse over event. Actions class’s object ‘builder’ instantiated in the above step, has a method known as ‘moveToElement’ which accepts the web element object to act upon (here link text). Using the ‘build’ method that return the Action object (here mouseoverevent) as shown

Action mouseoverevent = builder.moveToElement(headerelement).build();

mouseoverevent.perform();

* Use the Action class’s ‘perform’ method to move the mouse over the link element that was selected and passed as object in step above.
* Close the driver object and exit the system.