WAITS AND ACTIONS API

RAM SHARMA

WAITS INTRO

- Waiting is an automated step, elapse a certain amount of time, before execution can continue.
- Many websites are developed using Ajax and JavaScript. When a page is loaded by the browser the elements which we want to interact with may load at different time intervals.
- If the targeted element is not found, it will throw an "ElementNotVisibleException" exception. We can use waits to solve this problem.
- Types of Wait:
  - Implicit Waits
  - Explicit Waits
  - Static Wait : Thread.sleep(long timeInMillis)

IMPLICIT WAIT

- An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available.
- The default setting is 0 for implicit wait.
- The implicit wait is set for the life of the WebDriver object instance and hence applicable for every search.
- To set implicit wait:
  - driver.manage().timeouts().implicitlyWait(15, TimeUnit.SECONDS)
  - This specifies the amount of time the driver should wait when searching for an element if it is not immediately present.

EXPLICIT WAIT

- An explicit wait is code you define to wait for a certain condition to occur before proceeding further in the code.
- Worst form of Explicit Wait is static wait (sleep() method) in which it waits for entire specified time.
- We can achieve Explicit Wait by using below classes:
  - WebDriverWait
  - FluentWait
- Polling : It is a mechanism of repeatedly checking for existence of element after a fixed interval.
- You can define polling time using FluentWait where as WebDriverWait has polling time of 500 milliseconds.

EXPLICIT WAIT(CONTD.)

- until() method along with ExpectedConditions is used.
- Example:
  - WebElement myDynamicElement = (new WebDriverWait(driver, 10)).until(ExpectedConditions.presenceOfElementLocated(By.id("myDynamicElement")));
- static methods of ExpectedConditions:
  - static elementToBeClickable(WebElement element)
  - static elementToBeSelected(WebElement element)
  - static frameToBeAvailableAndSwitchToIt(WebElement frameLocator)
  - static presenceOfElementLocated(By locator)

ACTIONS INTRO

- Handling special keyboard and mouse events are done using the advanced user interactions API.
- Actions and Action interface is being used.
- We can add multiple actions in a single statement.
- perform() method starts performing the specified actions. It is present in both Actions and Action interface. It is only method of Action interface.
- Keyboard keys can be used using Keys interface.
  - Keys.CONTROL or Keys.SHIFT

ACTIONS API

- Action build()
- Actions click()
- Actions click(WebElement target)
- Actions clickAndHold()
- Actions clickAndHold(WebElement target)
- Actions contextClick()
- Actions contextClick(WebElement target)
- Actions doubleClick()
- Actions doubleClick(WebElement target)
- Actions dragAndDrop(WebElement source, WebElement target)
- Actions dragAndDropBy(WebElement source, int xCor, int yCor)

ACTIONS API(CONTD.)

- Actions keyDown(CharSequence key)
- Actions keyDown(WebElement target, CharSequence key)
- Actions keyUp(CharSequence key)
- Actions keyUp(WebElement target, CharSequence key)
- Actions moveByOffset(int xOffset, int yOffset)
- Actions moveToElement(WebElement target)
- Actions moveToElement(WebElement target, int xCor, int yCor)
- void perform()
- Actions release()
- Actions release(WebElement target)
- Actions sendKeys(CharSequence... keys)
- Actions sendKeys(WebElement target, CharSequence... keys)

AN EXAMPLE

- If the requirement is to perform mouse hover to menu and then submenu and click.
- WebElement menu;
- WebElement submenu;
- Actions actions = new Actions(driver);
- actions.moveToElement(menu).moveToElement(submenu).click() .perform();
- You can also build and perform as below:
  - actions.moveToElement(menu).moveToElement(submenu).click(). build().perform();