Pop-Up in Selenium & Parallel execution

In selenium depending on pop-up we write different types of code to perform action on the pop-up.

1. Alert or java-script pop-up
2. Hidden division or calendar pop-up
3. File upload pop-up
4. File download pop-up
5. Print pop-up
6. Notification pop-up
7. Authentication pop-up
8. Child window or Child browser pop-up

Alert pop-up-:

Character statics-:

- We cannot move this pop-up
- We cannot inspect this popup
- This pop-up will be having OK button (alert) or it contains OK and CANCLE button (Confirmation).
- It will be displayed below the address bar in the middle section of the browser.

Solution-:

To handle JavaScript popup we can us **driver.switchTo().alert()** statement.

- Accept()-: to click on OK button.
- Dismiss()-: to click on Cancle button.
- getText()-: to get the text present in the popup
- sendKeys()-: to enter/type the text on the popup

Example-:

```
            WebDriverWait wait = new WebDriverWait(driver,
Duration.ofSeconds(10));
            wait.until(ExpectedConditions.alertIsPresent());

            Alert action = driver.switchTo().alert();
            String text = action.getText();
            action.accept();
            System.out.println(text);
            driver.close();
```

Hidden division or calendar pop-up

Character static-:

- We can inspect the popup.
- We cannot move this popup.

Solution-: we can handle hidden division popup using driver.findElement().

```
driver.get("https://www.flipkart.com/");
driver.findElement(By.xpath("(//button)[2]")).click();
driver.close();
```

File download popup-:

Character static-:

- We can move this popup.
- We cannot inspect this popup.
- This popup includes "Open with" and "Save File" radio buttons, along with "OK" and "Cancel" buttons.

Solution-: To handle this popup we use Robot class.

Robot Class-:

- Robot class is a java class present in java.awt package. [Abstract window toolkit]
- Here we commonly use two methods keyPress and keyRelease.
- Robot class is mainly used to perform keyboard operation in windows.
- Robot class works almost similar to sendkeys() method.

Print Pop-up-:

Character statics-:

- We cannot move this popup.
- We cannot inspect this popup.
- This popup will be having print & cancel button.

Solution-: We handle this popup by using Robot class.

Notification popup-:

Character statics-:

- We cannot move this popup.
- We cannot inspect this popup.
- It will be having two buttons allow & block.
- It will be displayed below the address bar in the beginning similar to alert popup.

Solution-:

- To handle this popup we change the setting of the browser, so that notification popup itself will not be displayed.
- To change the setting of the browser we use **addArgument** method of **chromeOptions** class.

```
ChromeOptions option = new ChromeOptions();
option.addArguments("--disable-notifications");

ChromeDriver driver = new ChromeDriver(option);
driver.manage().window().maximize();
driver.get("https://www.yatra.com/");
```

Authentication Popup-:

Character statics-:

- We cannot move this popup.
- We cannot inspect this popup.
- This popup will be having username and password text box along with sign in and cancel button.

Solution-:

To handle authentication popup we send username and password along with the URL inside the get() method.

```
WebDriverManager.chromedriver().setup();
WebDriver driver = new ChromeDriver();
Thread.sleep(2000);
driver.get("https://admin:admin@the-
internet.herokuapp.com/basic_auth");
```

Child window or Child browser popup-:

Character statics-:

- We can move this popup.
- We can inspect this popup.
- This popup will be having minimize, maximize and close button along with the address bar.

Solution-:

To handle child browser popup we use **getWindowHandles()** and **driver.switchTo().window()** statement.

```
            Set<String> allWindow = driver.getWindowHandles();

            for (String window : allWindow) {
                  driver.switchTo().window(window);
                  String title = driver.getTitle();
                  System.out.println("The window title is : " +
title);
```

Parallel execution

To run a test script in parallel across 3 different browsers using Selenium WebDriver and TestNG, we can use the TestNG parallel execution feature.

Step 1: Create WebDriver Instances for Different Browsers

First, we need to create separate WebDriver instances for the browsers we want to run.

```
        {  WebDriver driver;
@Parameters("browser")
@Test
 public void testInMultipleBrowsers(String browser) {
        if (browser.equalsIgnoreCase("chrome")) {
            driver = new ChromeDriver();
          } else if (browser.equalsIgnoreCase("firefox")) {
            driver = new FirefoxDriver();
          } else if (browser.equalsIgnoreCase("edge")) {
            driver = new EdgeDriver(); }
```

```
        driver.get("http://yourtestsite.com");
        driver.quit();
```

Step 2: Configure TestNG XML for Parallel Execution

We need to create a testng.xml file and set it to run in parallel across the browsers.

We define different <parameter> tags for each browser.

```
<suite name="Parallel Browser Suite" parallel="tests" thread-count="3">
<test name="Chrome Test">
  <parameter name="browser" value="chrome" />
                <classes>
                  <class name="ParallelTest"/>
                </classes>
</test>
<test name="Firefox Test">
      <parameter name="browser" value="firefox" />
                <classes>
                  <class name="ParallelTest"/>
                </classes>
</test>
<test name="Edge Test">
       <parameter name="browser" value="edge" />
                <classes>
                  <class name="ParallelTest"/>
                </classes>
      </test>
    </suite>
```

Step 3: Run the Tests

When we run the testng.xml file, TestNG will execute the tests in parallel across Chrome, Firefox, and Edge.

Each test will run independently in its respective browser, and the thread-count defines the number of concurrent threads (in this case, 3 for 3 browsers).

Q. Handling AJAX popups using Selenium WebDriver:

1. **We can use Explicit Waits -:** Explicit Waits ensure that we are waiting for the element (popup) to appear before interacting with it.
2. **We can handle Stale Element Reference Exception-:** Stale Element Handling is crucial when dealing with dynamic content that can be reloaded asynchronously.
3. **Wait for the Ajax Call to Complete (via JavaScript)-:** JavaScript AJAX Checks allow for more control if AJAX requests might complete without immediately triggering a popup.

1. Use Explicit Waits

Since AJAX updates are asynchronous, the appearance of the popup might be delayed.

We can use explicit waits to wait for the popup to become visible or clickable.

```
// Trigger AJAX popup by clicking on some element
driver.findElement(By.id("triggerPopupButton")).click();
// Wait for the popup to appear
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement popup = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("ajaxPopup")));
popup.findElement(By.id("popupButton")).click();
driver.quit();
```

2. Handle Stale Element Reference Exception

In AJAX-based applications, sometimes elements are dynamically reloaded or replaced in the DOM, which can lead to StaleElementReferenceException.

We can handle this by catching the exception and retrying the operation.

```
try {
WebElement popup =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("ajaxPop
up")));
   popup.click();
} catch (StaleElementReferenceException e) {
   // Retry finding the element if it was replaced
   WebElement popup =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("ajaxPop
up")));
   popup.click();
```

3. Wait for the Ajax Call to Complete (via JavaScript)

In some cases, AJAX calls can be monitored by waiting for specific network activity or JavaScript events.

We can use JavaScript to check whether all AJAX requests have completed.

```
JavascriptExecutor js = (JavascriptExecutor) driver;
Boolean ajaxIsComplete = (Boolean) js.executeScript("return
jQuery.active == 0");
if (ajaxIsComplete) {
 driver.findElement(By.id("popupButton")).click();
```