Handling Web browser Automatically

Maximize the browser : You can't maximize firefox browser because this feature is inbuilt in geckodriver only. But if you want to maximize chrome browser then use the following line of code in your script.

```
driver.manage().window().maximize();
```

Navigating to any web application : For Navigating to any web Application we use a method called **get(String)** method or **navigate().to(String)** method present in webDriver.

```
driver.get(String);
driver.navigate().to(String);
```

Back, Forward and Refresh arrow in Browser : We can also play with back arrow , forward arrow and refresh arrow present in browser using methods available in webdriver.

```
driver.navigate().back();
driver.navigate().forward();
driver.navigate().refresh();
```

Changing Dimension of Browser : To change the Dimension of browser, we need to create an object of Dimension class present in Selenium package. Now we should invoke a method called **setSize(Dimension dim)** present in window class which takes the argument of type Dimension.

```
Dimension dim=new Dimension(width, height);
driver.manage().window().setSize(dim);
```

Changing Position of Browser : To change position of browser we need to create an object of Point class present in Selenium package. Now we should invoke a method called **setPosition(Point point)** present in window class which takes the argument of type Point.

```
Point point=new Point(width, height);
driver.manage().window().setPosition(point);
```

Printing Title of Web Application : getTitle() method in WebDriver is used to get the title of the web page or web application.

```
driver.getTitle();
```

Printing URL of the application : getCurrentUrl() method is used to get the current url of the application.

```
driver.getCurrentUrl();
```

Printing HTML/Source code of web Page : getPageSource() method can be used to get the HTML code or Source Code of the web page.

```
driver.getPageSource();
```

Closing the Web browser : When you are finished with the browser session you should call quit, instead of close. close() and quit() methods are used to close the browser.

```
driver.close();
driver.quit();
```

quit() will:

- Close all the windows and tabs associated with that WebDriver session
- Close the browser process
- Close the background driver process
- Notify Selenium Grid that the browser is no longer in use so it can be used by another session (if you are using Selenium Grid)
- Failure to call quit will leave extra background processes and ports running on your machine which could cause you problems later.

Handling Windows and Tabs : WebDriver does not make the distinction between windows and tabs. If your site opens a new tab or window, Selenium will let you work with it using a window handle. Each window has a unique identifier which remains persistent in a single session. You can get the window handle of the current window by using: **driver.getWindowHandle();**

Switching windows or Tabs : Clicking a link which opens in a new window will focus the new window or tab on screen, but WebDriver will not know which window the Operating System considers active. To work with the new window you will need to switch to it. If you have only two tabs or windows open, and you know which window you start with, by the process of elimination you can loop over both windows or tabs that WebDriver can see, and switch to the one which is not the original.However, Selenium 4 provides a new api **NewWindow** which creates a new tab (or) new window and automatically switches to it.

Create new window (or) new Tab and switch :Creates a new window (or) tab and will focus the new window or tab on screen. You don't need to switch to work with the new window (or) tab. If you have more than two windows (or) tabs opened other than the new window, you can loop over both windows or tabs that WebDriver can see, and switch to the one which is not the original.

Note: This feature works with Selenium 4 and later versions.

```
//Opens a new tab and switches to it
driver.switchTo().newWindow(WindowType.TAB);
```

```
// Opens a new window and switches to it
driver.switchTo().newWindow(WindowType.WINDOW);
```

Closing a window or Tab : When you are finished with a window or tab and it is not the last window or tab open in your browser, you should close it and switch back to the window you were using previously. Assuming you followed the code sample in the previous section you will have the previous window handle stored in a variable. Put this together and you will get:

```
//Close the tab or window
driver.close();
//Switch back to the old tab or window
driver.switchTo().window(originalWindow);
```

Forgetting to switch back to another window handle after closing a window will leave WebDriver executing on the now closed page, and will trigger a **No Such Window Exception**. You must switch back to a valid window handle in order to continue execution.

Questions :

1. **Write Test Script to validate Title and URL for any web application.**
2. **Write Test Script to print Title and URL for any web application.**
3. **Write Test Script to change the size of browser.**
4. **Write Test Script to displace the browser in some other location.**
5. **Write Test Script to maximize the browser.**
6. **Write Test Script to navigate to any application.**
7. **Write Test Script to close the browser.**

Frames and Iframes : Frames are a now deprecated means of building a site layout from multiple documents on the same domain. You are unlikely to work with them unless you are working with an pre HTML5 webapp. Iframes allow the insertion of a document from an entirely different domain, and are still commonly used.If you need to work with frames or iframes, WebDriver allows you to work with them in the same way. Consider a button within an iframe. If we inspect the element using the browser development tools, we might see the following:

```
<div id="modal">
  <iframe id="buttonframe" name="myframe"  src="https://seleniumhq.github.io">
   <button>Click here</button>
 </iframe>
</div>
```

If it was not for the iframe we would expect to click on the button using something like:

driver.findElement(By.tagName("button")).click(); //This won't work

However, if there are no buttons outside of the iframe, you might instead get a **no such element error**. This happens because Selenium is only aware of the elements in the top level document. To interact with the button, we will need to first switch to the frame, in a similar way to how we switch windows. WebDriver offers three ways of switching to a frame.

Switching to Frames/IFrames Using a WebElement : Switching using a WebElement is the most flexible option. You can find the frame using your preferred selector and switch to it.

```
WebElement iframe = driver.findElement(By.cssSelector("#modal>iframe"));     //Store the web element
driver.switchTo().frame(iframe);    //Switch to the frame
driver.findElement(By.tagName("button")).click();   //Now we can click the button
```

Switching to Frames/IFrames Using a name or ID : If your frame or iframe has an id or name attribute, this can be used instead. If the name or ID is not unique on the page, then the first one found will be switched to.

```
driver.switchTo().frame("buttonframe");  //Using the ID
driver.switchTo().frame("myframe");  //Or using the name instead
driver.findElement(By.tagName("button")).click();  //Now we can click the button
```

Switching to Frames/IFrames Using an index : It is also possible to use the index of the frame, such as can be queried using window.frames in JavaScript.

```
driver.switchTo().frame(1);   // Switches to the second frame
```

Leaving a frame : To leave an iframe or frameset, switch back to the default content like so:

```
driver.switchTo().defaultContent();        // Return to the top level
```

Window management : Screen resolution can impact how your web application renders, so WebDriver provides mechanisms for moving and resizing the browser window.

Get window size : Fetches the size of the browser window in pixels.

```
//Access each dimension individually
int width = driver.manage().window().getSize().getWidth();
int height = driver.manage().window().getSize().getHeight();
//Or store the dimensions and query them later
Dimension size = driver.manage().window().getSize();
int width1 = size.getWidth();
int height1 = size.getHeight();
```

Set window size : Restores the window and sets the window size

```
driver.manage().window().setSize(new Dimension(1024, 768));
```

Get window position : Fetches the coordinates of the top left coordinate of the browser window.

```
// Access each dimension individually
int x = driver.manage().window().getPosition().getX();
int y = driver.manage().window().getPosition().getY();
// Or store the dimensions and query them later
Point position = driver.manage().window().getPosition();
int x1 = position.getX();
int y1 = position.getY();
```

Set window position : Moves the window to the chosen position.

```
// Move the window to the top left of the primary monitor
driver.manage().window().setPosition(new Point(0, 0));
```

Maximise window : Enlarges the window. For most operating systems, the window will fill the screen, without blocking the operating system's own menus and toolbars.

```
driver.manage().window().maximize();
```

Minimize window : Minimizes the window of current browsing context. The exact behavior of this command is specific to individual window managers. Minimize Window typically hides the window in the system tray.

Note: This feature works with Selenium 4 and later versions.

```
driver.manage().window().minimize();
```

Fullscreen window : Fills the entire screen, similar to pressing F11 in most browsers.

```
driver.manage().window().fullscreen();
```

TakeScreenshot : Used to capture screenshot for current browsing context. The WebDriver endpoint screenshot returns screenshot which is encoded in Base64 format.

```
public class SeleniumTakeScreenshot {
    public static void main(String args[]) throws IOException {
        WebDriver driver = new ChromeDriver();
        driver.get("http://www.example.com");
        File scrFile =
            ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE)
            ;
        FileUtils.copyFile(scrFile, new File("./image.png"));
        driver.quit();
    }
}
```

TakeElementScreenshot : Used to capture screenshot of an element for current browsing context. The WebDriver endpoint screenshot returns screenshot which is encoded in Base64 format.

```
public class SeleniumElementTakeScreenshot {
  public static void main(String args[]) throws IOException {
    WebDriver driver = new ChromeDriver();
    driver.get("https://www.example.com");
                                        WebElement element =
driver.findElement(By.cssSelector("h1"));
    File scrFile = element.getScreenshotAs(OutputType.FILE);
    FileUtils.copyFile(scrFile, new File("./image.png"));
    driver.quit();
  }
}
```