Cheat sheet for Selenium Java

Hany Khalil

|  |  |
| --- | --- |
| **Setting up System Properties for browsers** | |
| **Chrome** | System.setProperty(“webdriver.chrome.driver”,  “/path/to/chromedriver”); |
| **Firefox** | System.setProperty(“webdriver.gecko.driver”,”/path/to/geckodriver”); |
| **IE** | System.setProperty(“webdriver.ie.driver", "Path/to/IEdriver "); |

|  |
| --- |
| **Safari** |
| * Run the following command from the terminal for the first time and type your password at the prompt to authorise WebDriver   safaridriver --enable   * Enable the Developer menu from Safari preferences * Check the Allow Remote Automation option from with the Develop menu * Run the following command from the terminal for the first time and type your password at the prompt to authorise WebDriver   /usr/bin/safaridriver -p 1337< |

|  |  |
| --- | --- |
| **Initializing the driver** | |
| **Chrome** | WebDriver driver = new ChromeDriver(); |
| **Firefox** | WebDriver driver = new FirefoxDriver(); |
| **IE** | WebDriver driver = new InternetExplorerDriver(); |
| **Safari** | WebDriver driver = new SafariDriver(); |

|  |  |
| --- | --- |
| **WebDriverManager** | |
| **Chrome** | WebDriverManager.chromedriver().setup(); |
| **Firefox** | WebDriverManager.firefoxdriver().setup(); |
| **IE** | WebDriverManager.iedriver().setup(); |

|  |  |
| --- | --- |
| **Browser Navigation** | |
| **Navigate to** | driver.get("https://selenium.dev"); driver.navigate().to("https://selenium.dev"); |
| **Get current URL** | driver.getCurrentUrl(); |
| **Back** | driver.navigate().back(); |
| **Forward** | driver.navigate().forward(); |
| **Refresh** | driver.navigate().refresh(); |

|  |  |
| --- | --- |
| **Handle Windows and tabs** | |
| **Get window handle of the current window** | driver.getWindowHandle(); |
| **Get handles of all the**  **windows opened** | driver.getWindlowHandles(); |
| **Open a new tab and switches to the new tab** | driver.switchTo().newWindow(WindowType.TAB); |
| **Open a new window and switches to new window** | driver.switchTo().newWindow(WindowType. WINDOW); |
| **Switch back to old tab or window** | driver.switchTo().window(originalWindow); |
| **Closed the current browser window** | driver.close(); |
| **Close all the windows and**  **tabs associated with the Webdriver session** | driver.quit(); |

|  |  |
| --- | --- |
| **Element Validation** | |
| **isEnabled** | driver.findElement(By.name("btnK")).isEnabled(); |
| **isSelected** | driver.findElement(By.name("yes")). isSelected(); |
| **isDisplayed** | driver.findElement(By.name("btnK")).isDisplayed(); |

|  |  |
| --- | --- |
| **Handling Frames** | |
| **Switch to specific frame with the help of Webelement.** | driver.switchTo().frame(webElement iframe); |
| **Switch to specific frame with the help of frame name** | driver.switchTo().frame(string frameName); |
| **Switch to specific frame with the help of frame ID** | driver.switchTo().frame(string ID); |
| **Switch to specific frame with the help of index** | driver.switchTo().frame(int frameNumber); |
| **Switch back to the main window** | driver.switchTo().defaultContent(); |

|  |  |
| --- | --- |
| **Window Management** | |
| **To get the size of the browser window** | driver.manage().window().getSize.getWidth();  driver.manage().window().getSize.getHeight(); |
| **To restore the window and sets the window size** | driver.manage().window().setSize(new Dimension(1024, 768)); |
| **To get the coordinates of the browser window** | driver.manage().window().getPosition().getX();  driver.manage().window().getPosition().getY(); |
| **To set the window position x=100 and y=200** | driver.manage().window().setPosition(new  Point(100, 300)); |
| **To maximize the browser**  **window** | driver.manage().window().maximize(); |
| **To minimize the browser**  **window** | driver.manage().window().minimize(); |
| **To enlarge the browser and fills the entire screen, similar**  **to pressing F11 in most browsers.** | driver.manage().window().fullscreen(); |

|  |  |
| --- | --- |
| **Handling Alerts** | |
| **To capture the alert message.** | driver.switchTo().alert.getText(); |
| **Click 'OK' button on the alert.** | driver.switchTo().alert.accept(); |
| **Click 'Cancel' button on the alert.** | driver.switchTo().alert.dismiss(); |
| **To send some data to alert box.** | alert.sendKeys("Selenium"); |

|  |  |
| --- | --- |
| **Get Commands** | |
| To get the title of currently opened web page | driver.getTitle(); |
| To get the URL of the currently opened web page | driver.getCurrentUrl(); |
| To get the Page source of currently opened web page | driver.getPageSource(); |
| To get the text of a webElement | driver.findElement(By.name("q")).getText(); |
| To get the value of the web element’s attribute | driver.findElement(By.name("q")).getAttribute("type"); |

|  |  |
| --- | --- |
| **Locators** | |
| **By ID** | Locates the element whose ID attribute matches the search value  driver.findElement(By.id(“email”); |
| **By name** | Locates elements whose NAME attribute matches the search value  driver.findElement(By.name(“email”); |
| **By Cssselector** | Locates elements matching a CSS selector  driver.FindElement(By.cssSelector("#email")); |
| **By LinkText** | Locates anchor elements whose visible text matches the search value  driver.FindElement(By.linkText("Forgotten password? ")); |
| **By PartialLinkText** | Locates anchor elements whose visible text contains the search value  driver.FindElement(By.partialLinkText("password?")); |
| **By Tagname** | Locates elements whose tag name matches the search value  driver.FindElement(By.tagName("a")); |
| **By class name** | Locates elements whose class name contains the search value  driver.FindElement(By.className("email")); |
| **By Xpath** | Locates elements matching an XPath expression  driver.FindElement(By.XPath(“//input[@id='email'] "); |
| **above()** | Locates the element, which appears above to the specified element  driver.findElement(with(By.tagName("input")) .above(passwordField)); |
| **below()** | Locates the element which appears below to the specified element  driver.findElement(with(By.tagName("input")) .below(emailAddressField)); |
| **toLeftOf()** | Locates the element which appears to left of the specified element  driver.findElement(with(By.tagName("button")) .toLeftOf(submitButton)); |
| **toRightOf()** | Locates the element which appears to left of the specified element  driver.findElement(with(By.tagName("button")).toRightOf(submitButton)); |
| **near()** | Locates the element which is at most 50px away from the specified element.  driver.findElement(with(By.tagName("input")).near(emailAddressLabel)); |

|  |  |
| --- | --- |
| **Keyboard events** | |
| **SendKeys()** | driver.findElement(By.name("q")).sendKeys("q"); |
| **clear()** | driver.findElement(By.name("q")).clear(); |
| **keyUp()** | Actions action = new Actions(driver); action.keyDown(Keys.CONTROL); |
| **keyDown()** | Actions action = new Actions(driver); action.keyUp(Keys.CONTROL); |

|  |  |
| --- | --- |
| **Mouse Events** | |
| **clickAndHold** | Actions action = new Actions(driver);  action.clickAndHold(webElement).build().perform(); |
| **contextClick** | Actions action = new Actions(driver);  action.contextClick(webElement).build().perform(); |
| **doubleClick** | Actions action = new Actions(driver);  action.doubleClick(webElement).build().perform(); |
| **moveToElement** | Actions action = new Actions(driver);  action.moveToElement(webElement).build().perform(); |
| **moveByOffset** | Actions action = new Actions(driver);  action.moveByOffset(xOffset,yOffset).build().perform(); |
| **dragAndDrop** | Actions action = new Actions(driver);  action.dragAndDrop(sourceEle,targetEle).build().perform(); |
| **dragAndDropBy** | Actions action = new Actions(driver); action.dragAndDropBy(sourceEle, targetEleXOffset,  targetEleYOffset).build().perform(); |
| **release** | Actions action = new Actions(driver);  action.release().build().perform(); |

|  |  |
| --- | --- |
| **Dropdowns** | |
| Select an option from the dropdown based on index | selectByIndex(1); |
| Select an option from the dropdown based on its value attribute | selectByValue("value1"); |
| Select an option from the dropdown based on the visible text | selectByVisibleText("White"); |
| To clear the selected entries of the dropdown | deselectAll(); |
| Deselect an option from  the dropdown based on index | deselectByIndex(5); |
| Deselect an option from  the dropdown based on its value attribute | deselectByValue(“value1”); |
| Deselect an option from  the dropdown based on the visible text | deselectByVisibleText(“White”); |
| To get all the options in a dropdown or multi-select box | getOptions(); |
| To get the first selected option of the dropdown. | getFirstSelectedOption(); |
| To get all the selected options of the dropdown | getAllSelectedOptions(); |

|  |  |
| --- | --- |
| **WebElements** | |
| **FindElement** | It returns first matching single WebElement reference  When no match has found(0 elements) throws NoSuchElementException  Syntax:  WebElement findElement(By by) |
| **FindElements** | It returns a list of all matching WebElements  When no match has found(0 elements) throws emptyLis- tofWebElementObject  Syntax:  List<WebElement> findElements(By by) |
| **Find Element From Element** | It is used to find a child element within the context of parent element  Syntax:  WebElement searchForm  =driver.findElement(By.tagName("form")); WebElement searchBox = searchForm.findElement(By.name("q")); |
| **Find Elements From Element** | It is used to find the list of matching child WebElements within the context of parent element  Syntax:  WebElement element = driver.findElement(By.tagName("div")); List<WebElement> elements = element.findElements(By.tagName("p")); |
|  |  |
| **Cookies** | |
| **Add Cookie** | driver.manage().addCookie(new Cookie("key", "value")); |
| **Get Named Cookie** | driver.manage().getCookieNamed("foo"); |
| **Get All Cookies** | driver.manage().getCookies(); |
| **Delete Cookie** | driver.manage().deleteCookie(cookie1); |
| **Delete All Cookies** | driver.manage().deleteAllCookies(); |

|  |
| --- |
| **Waits** |
| **Implicit wait** – An implicit wait is to tell the 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  driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); |
| **Fluent Wait** – It defines the maximum amount of time to wait for a certain condition as well as the frequency to check for the condition to appear  Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)  .withTimeout(Duration.ofSeconds(30))  .pollingEvery(Duration.ofSeconds(5))  .ignoring(NoSuchElementException.class);  WebElement foo = wait.until(new Function<WebDriver, WebElement>() { public WebElement apply(WebDriver driver) {  return driver.findElement(By.id("foo"));  }  }); |
| **Explicit Wait** – It is used to wait until a certain condition occurs before proceeding further in the code.  WebDriverWait wait = new WebDriverWait(driver,30); wait.until(ExpectedConditions.presenceOfElementLocated(By.name("login"))); |

|  |
| --- |
| **Screenshots** |
| **TakeScreenshot:**  It is used to capture screenshot for current browsing context  File scrFile = ((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE); FileUtils.copyFile(scrFile, new File("./image.png")); |
| **TakeElementScreenshot:**  It is Used to capture screenshot of an element for current browsing context.  File scrFile = element.getScreenshotAs(OutputType.FILE); FileUtils.copyFile(scrFile, new File("./image.png")); |