testRigor

Selenium with Java Cheat Sheet

Locating Elements

By ID:
```
driver.findElement(By.id (<element ID>))
```

By Name:
```
driver.findElement(By.name(<element name>))
```

By Class Name:
```
driver.findElement(By.className (<element class>))
```

By Tag Name:
```
driver.findElement(By.tagName (<html tag name>))
```

By CSS Selector:
```
driver.findElement(By.cssSelector("Tag#Value of id attribute"))
```

By XPath:
```
driver.findElement(By.xpath (“//input[@type='submit']”))
```

By Link Text:
```
driver.findElement(By.linkText (<link text>))
```

By Partial Link Text:
```
driver.findElement(By.partialLinkText (<link text>))
```

Working with Windows

Get the current window handle:
```
String mainWindowHandle = driver.getWindowHandle();
```

Get all window handles:
```
import java.util.Set;
Set<String> allWindowHandles = driver.getWindowHandles();
```

Switch to a specific window:
```
String windowHandle = driver.getWindowHandle();
driver.switchTo().window(windowHandle);
```

Switch to newly created window:
```
driver.switchTo().newWindow(WindowType.TAB);
driver.switchTo().newWindow(WindowType.WINDOW);
```

Close the current window:
```
driver.close();
```

Set window position:
```
driver.manage().window().setPosition(new Point(0, 0));
```

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

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

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

Take a Screenshot:
```
import org.apache.commons.io.FileUtils;
File scrFile =
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
FileUtils.copyFile(scrFile, new File("./image.png"));
```

Working with Frames

Switch to a frame by name or ID:
```
driver.switchTo().frame("buttonframe");
```

Switch to a frame by index:
```
driver.switchTo().frame(1);
```

Switch to a frame using a WebElement:
```
WebElement iframe =
driver.findElement(By.cssSelector("#modal>iframe"));
driver.switchTo().frame(iframe);
```

Switch back to the main content:
```
driver.switchTo().defaultContent();
```

Working with Files

Upload a file:
```
driver.findElement(By.id("file-
upload")).sendKeys(“path/to/your/file.txt”);
driver.findElement(By.id("file-submit")).submit();
```

Read data from a text file - Using BufferedReader:
```
FileInputStream inputStream = new FileInputStream(“MyFile.txt”);
InputStreamReader reader = new InputStreamReader(inputStream, “UTF-
16”);
int character;
while ((character = reader.read()) != -1) {
  System.out.print((char) character);
}
reader.close();
```

Using FileReader:
```
FileReader reader = new FileReader(“MyFile.txt”);
int character;
while ((character = reader.read()) != -1) {
  System.out.print((char) character);
}
reader.close();
```

Read data from a CSV file:
```
mport au.com.bytecode.opencsv.CSVReader;
String path = “C:\\Users\\Myuser\\Desktop\\csvtest.csv”;
Reader reader = new FileReader(path);
CSVReader csvreader = new CSVReader(reader);
List<String[]> data = csvreader.readAll();
for(String[] d : data){
  for(String c : d ){
    System.out.println(c);
  }
}
```

Read data from an Excel file:
```
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import java.io.File;
import java.io.FileInputStream;
import java.io.IOException;
File file = new File(“E:\\TestData\\TestData.xls”);
FileInputStream inputStream = new FileInputStream(file);
HSSFWorkbook wb=new HSSFWorkbook(inputStream);
HSSFSheet sheet=wb.getSheet(“STUDENT_DATA”);
HSSFRow row2=sheet.getRow(1);
HSSFCell cell=row2.getCell(5);
String address= cell.getStringCellValue();
```

Driver Initialization

Chrome
```
WebDriver driver = new ChromeDriver();
```

Firefox
```
WebDriver driver = new FirefoxDriver();
```

Edge
```
WebDriver driver = new EdgeDriver();
```

Safari
```
WebDriver driver = new SafariDriver();
```

Selenium Grid

Start the hub

```
java -jar selenium-server-standalone-x.y.z.jar -role hub
```

Start a node

```
java -jar selenium-server-standalone-x.y.z.jar -role node -hub
http://localhost:4444/grid/register
```

Server

```
http://localhost:4444/ui/index.html
```

Note: Replace "x.y.z" with the version number of your Selenium Server Standalone JAR file.

Selenium Grid

Start the hub

```
java -jar selenium-server-standalone-x.y.z.jar -role hub
```

Start a node

```
java -jar selenium-server-standalone-x.y.z.jar -role node -hub
```

Server

```
http://localhost:4444/ui/index.html
```

Selenium Navigators

Navigate to a URL	`driver.get("<URL>") or driver.navigate().to("<URL>")`
Refresh the page	`driver.navigate().refresh()`
Navigate forward in browser history	`driver.navigate().forward()`
Navigate back in browser history	`driver.navigate().back()`

Mouse Actions

Click:

```
var actions = new Actions(driver);
actions.Click(element).Perform();
```

Double click:

```
actions.DoubleClick(element).Perform();
```

Right click:

```
actions.ContextClick(element).Perform();
```

Drag and drop:

```
actions.DragAndDrop(sourceElement, targetElement).Perform();
```

Move to element:

```
actions.MoveToElement(element).Perform();
```

JUnit

@Test	Represents the method or class as a test block, also accepts parameters.
@Before	The method with this annotation gets executed before all the other tests.
@BeforeClass	The method with this annotation gets executed once before class.
@After	Will execute before each test method runs
@AfterClass	The method with this annotation gets executed after all the other tests are executed.
@Ignore	The method with this annotation gets executed once after class.
@Disabled	Used to disable the tests from execution, but the corresponding reports of the tests are still generated.

Working with Alerts

Switch to an alert:

```
Alert alert = driver.switchTo().alert();
```

Enter text in an alert:

```
alert.sendKeys("Selenium");
```

Retrieve alert text:

```
String text = alert.getText();
```

Selenium Operations

Launch a Webpage:

```
driver.get("<URL>") or driver.navigate().to("<URL>")
```

Click a button:

```
WebElement searchButton = driver.findElement(By.name("btnK"));
searchButton.click();
```

Print the page title:

```
String title = driver.getTitle();
System.out.println(title);
```

Accept an alert pop-up:

```
driver.switchTo( ).alert( ).accept();
```

Implicit wait:

```
import java.util.concurrent.TimeUnit;
driver.manage().timeouts().implicitlyWait(30, TimeUnit.SECONDS);
```

Explicit wait:

```
import java.util.concurrent.TimeUnit;

WebElement firstResult = new WebDriverWait(driver,
Duration.ofSeconds(10))

 .until(ExpectedConditions.elementToBeClickable(By.xpath("//a/h3")));
```

Sleep:

```
Thread.sleep(<Time in MilliSeconds>);
```

Clear the input field text:

```
WebElement searchInput = driver.findElement(By.name("q"));
searchInput.sendKeys("selenium");
searchInput.clear();
```

Disable a field (set the 'disabled' attribute):

```
((IJavaScriptExecutor)driver).ExecuteScript("document.querySelector('
<element_css_selector>').setAttribute('disabled', '');");
```

Enable a field (remove the 'disabled' attribute):

```
((IJavaScriptExecutor)driver).ExecuteScript("document.querySelector('
<element_css_selector>').removeAttribute('disabled');");
```

TestNG

@BeforeSuite	Will run before the execution of all the test methods in the suite
@BeforeTest	Will execute before the execution of all the test methods of available classes belonging to that folder
@BeforeClass	Will execute before the first method of the current class is invoked
@BeforeMethod	Will execute before each test method runs
@Test	This is the main part of our automation script where we write the business logic we want to automate
@AfterMethod	Will execute after the execution of each test method
@AfterClass	Will execute after the execution of all the test methods of the current class
@AfterTest	Will execute after the execution of all the test methods of available classes belonging to that folder
@AfterSuite	Will execute after the execution of all the test methods in the suite