A quick guide to - Selenide

Selenide is a framework for test automation (wrapper on the selenium web driver)

Website: https://www.selenide.org

Features of Selenide:

1. Concise fluent API for tests
2. Ajax support
3. Powerful selectors
4. Simple configuration
5. Handles browser's → selection, shutdown, timeouts
6. Handles web element's → timeout, Stale Element Exceptions
7. Handles test's→ search for relevant logs, debugging
8. Natural language assertions

Maven Dependency:

```
<dependency>
    <groupId>com.codeborne</groupId>
    <artifactId>selenide</artifactId>
    <version>6.4.0</version>
    <scope>test</scope>
</dependency>
```

(**Note**- find latest version at https://mvnrepository.com/artifact/com.codeborne/selenide)

Static imports required to quickstart:

```
import static com.codeborne.selenide.Selenide.*;
import static com.codeborne.selenide.Condition.*;
```

Browser Actions:

1. **`open("web URL here")`** --> Open desired url (without any credentials)
OR **open (url, domain, loginName, password)** --> Open desired url (with certain credentials)
2. **`title()`** --> to get the title of page as a string
3. **`back()`** --> to navigate back to previous page
4. **`forward()`** --> to navigate forward to page whose url is already hit earlier
5. **`refresh()`** --> to refresh current page
6. **`sleep(desired timeout in milliseconds)`** --> alternative to Thread.sleep()
7. **`screenshot("filename")`** --> to take screenshot of open page

8. **`closeWindow()`** --> alternative to driver.close() to close current window
9. **`closeWebDriver()`** --> alternative to driver.quite() to close all window & kill driver

---

Locators:

Note: $ → to locate single element, $$ → to locate array of elements
1. **`$(By.name("name attribute vale here"))`** → to locate single element by name
2. **`$(By.id("id attribute value here"))`** → to locate single element by id
3. **`$(By.xpath("xpath value here"))`** → to locate single element by xpath
OR **`$x("xpath value here")`**
4. **`$(By.partialLinkText("partial link text here"))`** → to locate single element by partial link text
5. **`$$(By.cssSelector("css value here"))`** → to locate array of element by css
6. **`$$(By.xpath("xpath value here"))`** → to locate array of element by xpath
OR **`$$x ("xpath value here")`**

Element action methods:

1. **`setValue("desired value")`** → to set desired value in text box element located (faster)
2. **`sendKeys("desired value")`** → to set desired value in text box element located (slower)
3. **`click()`** → to perform click action on located element
4. **`getText()`** → to get the text from the located element
5. **getAttribute()** → to get the value of specific attribute out of located element

---

Assertions:

1. **`should(Condition/CollectionCondition.<desired condition/s>)`**
OR **`should(Condition/CollectionCondition.<desired condition/s>, Duration timeout)`**
2.**`shouldHave(Condition/CollectionCondition.<desired condition/s>)`**
OR **`shouldHave(Condition/CollectionCondition.<desired condition/s>, Duration timeout)`**
3. **`shouldNot(Condition/CollectionCondition.<desired condition/s>)`**
OR **`shouldNot(Condition/CollectionCondition.<desired condition/s>, Duration timeout)`**
4. **`shouldNotBe(Condition/CollectionCondition.<desired condition/s>)`**
OR **`shouldNotBe(Condition/CollectionCondition.<desired condition/s>, Duration timeout)`**
5. **`shouldNotHave(Condition/CollectionCondition.<desired condition/s>)`**
OR **`shouldNotHave(Condition/CollectionCondition.<desired condition/s>, Duration timeout)`**

Actions on collection of elements: (located by $$)

1. **`forEach(CollectionCondition.desired collection condition OR lambda expression)`** → to perform specific operation on each element out of the collection
2**`. stream().filter(CollectionCondition.desired collection condition OR lambda expression).forEach(CollectionCondition.desired collection condition OR lambda expression)`** → filter out the collection to perform specific operation on each of the remained
3. **`last(int number).forEach(CollectionCondition.desired collection condition OR lambda expression)`** → perform specific operation on each of the last mentioned number of elements out of the collection

4. **first(int number).forEach(CollectionCondition.desired collection condition OR lambda expression)** → perform specific operation on each of the first mentioned number of elements out of the collection
5. **subList(fromIndex, toIndex).forEach(CollectionCondition.desired collection condition OR lambda expression)** → perform specific operation on each of the mentioned range of elements out of the collection
6. **texts()** → to get list of texts from each of the element out of the collection

---

Selenide Configuraion:

Class Name: `Configuraiton`
Import required: `import com.codeborne.selenide.Configuration;`
Methods:

1. **browser = "desired browser name"** → to set desired browser as test browser
   OR **System.setProperty("selenide.browser", "desired browser name")** → using system property
   OR **mvn clean install -Dselenide.browser = "desired browser name"** → using maven
2. **headless = true** → to run browser in headless mode
3. **baseUrl = "desired url"** → to set base url to be tested
4. **browserBinary = "desired absolute path of driver"** → to set the path of browser driver
   OR **System.setProperty("selenide.browserBinary", "desired absolute path of driver")**
5. **startMaximized = true** → launch the browser in maximised mode
6. **screenshots = true** → to take screenshot of failed screenshot at /build/reports/
7. **browserPosition**
8. **browserSize**
9. **browserVersion**
10. **browserCapabilities**
11. **clickViaJs**
12. **downloadsFolder**
13. **driverManagerEnabled**
14. **fastSetValue**
15. **fileDownload**
16. **pageLoadStrategy**
17. **pageLoadTimeout**
18. **pollingInterval**
19. **proxyEnabled**
20. **proxyHost**
21. **proxyPort**
22. **remote**
23. **reopenBrowserOnFail**
24. **reportsFolder**
25. **reportsUrl**
26. **savePageSource**
27. **selectorMode**
28. **timeout**
29. **webdriverLogsEnabled**
30. **assertionMode**
31. **holdBrowserOpen = True** → to hold browser open until all test classes are executed

---

Runner Class:

Class name: `WebDriverRunner`

methods:

1. **`clearBrowserCache()`** → to delete cache of browser under test
2. **`url()`** → alternative to driver.getUrl(), to get current url of page
3. **`source()`** → alternative to driver.getPageSource(), to get the HTML page source
4. **`closeWebDriver()`** → alternative to driver.quite() to close all window & kill driver
5. **`closeWindow()`** → alternative to driver.close() to close current window
6. **`getWebDriver()`** → get the webdriver control from selenide to selenium
7. **`isChrome(), isFirefox(), isLegacyFirefox(), isEdge(), isIE(), isOpera(), isSafari()`**
8. **`isHeadless()`** → to check if browser is running in headless mode
9. **`supportJavascript()`** → to check if current browser is supporting JavaScript
10. **`currentFrameUrl()`** → url of current frame under test
11. **`getBrowserDownloadsFolder()`** → to get the location of download folder associated with browser
    a. **`cleanupBeforeDownload()`** → to clean all contents of download folder before next download
    b. **`file(String fileName)`** → to get specific file from download folder
    c. **`files()`** → to get list of files from download folder
    d. **`toFile()`**
    e. **`toString()`** → to get path of download folder as a string

Handling JavaScript popups: (The popup where user cannot inspect anything)

`switchTo().alert()`

1. **`getText()`**
2. **`accept()`**
3. **`dismiss()`**
4. **`sendKey(String prompt)`**

Handling iframes:

`switchTo().`

1. **`frame(int index)`** → to switch to DOM of specific iframe
   OR **`frame(String name/id)`**
   OR **`frame(WebElement frameElement)`**
2. **`innerFrame(String firstFrame, String otherFrames....)`**
3. **`parentFrame()`** → to switch to DOM of parent frame
4. **`defaultContent()`** → to switch to main content of the page containing iframe

---

Handling Authentication popups: (another popup where user cannot inspect anything)

`open (URL/String url, domain, loginName, password)` --> not working for safari

Handling 'Select' tag based dropdown:

1. **`selectOption(int index)`** → select option by index of selection
   OR **`selectOption(String text)`** → select option by visible text of selection
2. **`selectOptionByValue(String value)`** → select option by value of selection
2. **`selectOptionContainingText(String text)`** → select option by parial text value of selection

3. **`getSelectedOption()`** → get String value of first selected option
4. **`getSelectedOptions()`** → get List of String values of selected options

Handling non select based dropdown:

```
ElementsCollection coll = $$x("Desired xpath");
for(SelenideElement e : coll){
      String text = e.getText();
      if(text.equals("desired option")) {
            e.click();
            break;
      }
}
```

User Actions: alternative to actions class (to perform mouse/keyboard actions)

Method: `actions()`
Chained methods:

1. **. **`moveToElement(WebElement target)`** → move cursor to certain element
   OR **`moveToElement(WebElement target, int xOffset, int yOffset)`**
   → move cursor to certain element and then by X-Y offsets
2. **`moveByOffset(int xOffset, int yOffset)`** → move cursor to by X-Y offsets from current location
3. **`click()`**
   OR **`click(WebElement target)`**
5. **`clickAndHold()`**
   OR **`clickAndHold(WebElement target)`**
6. **`contextClick()`** → perform right click
   OR **`contextClick(WebElement target)`** → perform right click on specific element
7. **`doubleClick()`** → perform double click
   OR **`doubleClick(WebElement target)`** → perform double click on specific element
8. **`dragAndDrop(WebElement source, WebElement target)`** → drag one element onto another
   OR **`dragAndDrop(WebElement source, int xOffset, int yOffset)`**
9. **`release(WebElement target)`**
10. **`sendKeys(CharSequence...keys)`**
    OR **`sendKeys(WebElement target, CharSequence...keys)`**
4. **`tick(Action action)`**
   OR **`tick(Interaction... actions)`**
11. **`keyDown(CharSequence key)`**
    OR **`keyDown(WebElement target, CharSequence key)`**
12. **`keyUp(CharSequence key)`**
    OR **`keyUp(WebElement target, CharSequence key)`**
13. **`pause(Duration duraiton)`**
    OR **`pause(long duration)`**
14. **`notify()`**
5. OR **`notifyAll()`**
15. **`wait()`**
    OR **`wait(long timeout)`**
    OR **`wait(long timeout, int nanos)`**

16. ** **`build()`** → bind multiple actions together
17. ** **`perform()`** → perform specified actions in given order

** *mandatory after each sequence of user action (mouse/keyboard)*

Waits in Selenide:

Important note: *Default explicit wait in selenide = 4 sec*
To set default timeout: `Configuration.timeout = desired timeout in milliseconds`
Syntax: `SelenideElement.shouldxxx(Condition.yyy, Duration.ofzzz)`
Available assertions:

1. **should**
   OR shouldBe
2. **shouldHave**
3. **shouldNot**
4. **shouldNotBe**
6. **shouldNotHave**

Available conditions:

Class: `Condition`
Methods:

1. **appear**
2. **appears**
3. **checked**
4. **disabled**
5. **disappear**
6. **empty**
7. **enabled**
8. **exist**
9. **focused**
10. **hidden**
11. **image**
12. **readonly**
13. **selected**
14. **visible**
15. **and** → for multiple conditions

Available Durations:

Class: `Duration`
methods:

1. **of(long amount, TemporalUnit unit)**
2. **ofDays(long days)**
3. **ofHours(long hours)**
4. **ofMinutes(long minutes)**
5. **ofSeconds(long seconds)**
   OR **ofSeconds(long seconds, long nanaoAdjustment)**
6. **ofMillis(long millis)**
7. **ofNanos(long nanos)**
