Selenium WebDriver Java Cheatsheet

A quick reference guide for essential Selenium WebDriver commands, locators, waits, and patterns in Java.

Setup & Basics

Maven Setup (pom.xml)

Add Selenium Java dependency:

```
<dependency>

<groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.18.1</version>
</dependency>
```

Add WebDriverManager (optional, but recommended):

```
<dependency>

<groupId>io.github.bonigarcia</groupId>

<artifactId>webdrivermanager</artifactId>
    <version>5.8.0</version>
</dependency>
```

Add TestNG/JUnit dependency:

```
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.9.0</version>
    <scope>test</scope>
</dependency>
```

(Or JUnit)

Basic project structure:

- `src/main/java` (for Page Objects, utilities)
- `src/test/java` (for test classes)
- `src/test/resources` (for test data, config)

Create a new Maven project in your IDE (Eclipse, IntelliJ).

Update Maven project (`Maven -> Update Project...`).

Ensure Java Development Kit (JDK) is installed and configured.

WebDriver Initialization

Basic Initialization:

```
// Manual driver setup (requires driver executable path)
System.setProperty("webdriver.chrome.driver", "/path/to/chromedriver");
WebDriver driver = new ChromeDriver();
```

Using WebDriverManager (Recommended):

```
// Automatically downloads & sets up driver
WebDriverManager.chromedriver().setup();
WebDriver driver = new ChromeDriver();
```

Other Browsers:

```
WebDriverManager.firefoxdriver().setup();
WebDriver driver = new FirefoxDriver();

WebDriverManager.edgedriver().setup();
WebDriver driver = new EdgeDriver();

WebDriverManager.safaridriver().setup();
// Requires Safari 10+
WebDriver driver = new SafariDriver();
```

Configure Browser Options:

```
ChromeOptions options = new ChromeOptions();
options.addArguments("--headless"); // Example: Run in headless mode
WebDriver driver = new ChromeDriver(options);
```

Maximize Window:

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

Set Implicit Wait (Not recommended with Explicit Waits):

```
driver.manage().timeouts().implicitlyWait(java.time.Duration.ofSeconds(10));
```

Basic Browser Commands

Navigate to URL

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

Navigate Back

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

Navigate Forward

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

Refresh Page

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

Get Current URL

```
String currentUrl = driver.getCurrentUrl();
```

Get Page Title

```
String pageTitle = driver.getTitle();
```

Get Page Source

```
String pageSource = driver.getPageSource();
```

Close Current Window

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

Quit Browser Session

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

Locators & Elements

Element Locators (By Class)

`By.id("...")`

Locate element by its ID attribute.
Example: `By.id("loginButton")`

`By.name("...")`

Locate element by its Name attribute.
Example: `By.name("username")`

`By.className("...")`

Locate element by its CSS class name.
Example: `By.className("form-input")`
Note: Compound classes not allowed (use CSS Selector)

`By.tagName("...")`

Locate elements by their HTML tag name.
Example: `By.tagName("a")`

`By.linkText("...")`

Locate anchor elements (`<a>`) by their full text.
Example: `By.linkText("Click Here")`

`By.partialLinkText("...")`

Locate anchor elements (`<a>`) by partial text match.
Example: `By.partialLinkText("Click")`

`By.cssSelector("...")`

Locate elements using CSS Selectors (flexible & fast).
Example:
`By.cssSelector("input[type='text']")`

`By.xpath("...")`

Locate elements using XPath expressions (powerful, but potentially brittle).
Example:
`By.xpath("//button[@id='submit']")`

Finding Elements

Find a single element:

```
WebElement element =
driver.findElement(By.id("myElement"));
```

Throws `NoSuchElementException` *if not found.*

Find multiple elements:

```
List<WebElement> elements =
driver.findElements(By.tagName("a"));
```

Returns an empty list if no elements are found (does not throw exception).

Find element within another element:

```
WebElement parent =
driver.findElement(By.id("parentElement"
));
WebElement child =
parent.findElement(By.className("childEl
ement"));
```

Find multiple elements within another element:

```
WebElement parent =
driver.findElement(By.id("parentElement"
));
List<WebElement> children =
parent.findElements(By.tagName("li"));
```

Working with Elements

Click an element

```
element.click();
```

Type text into input field

```
WebElement input =
driver.findElement(By.id("username"));
input.sendKeys("myuser");
```

Clear text from input field

```
input.clear();
```

Submit a form

```
WebElement form =
driver.findElement(By.id("loginForm"));
form.submit();
// Or submit using an element within the
form:
// element.submit();
```

Get element text

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

Get attribute value

```
String attribute =
element.getAttribute("value");
```

Get CSS property value

```
String cssValue =
element.getCssValue("color");
```

Check if element is displayed

```
boolean isDisplayed =
element.isDisplayed();
```

Check if element is enabled

```
boolean isEnabled = element.isEnabled();
```

Check if element is selected (checkbox/radio)

```
boolean isSelected =
element.isSelected();
```

Waits & Synchronization

Implicit Wait

Sets a default timeout for finding *any* element.
Applies globally to all `driver.findElement` calls.
Less flexible than Explicit Waits, can slow down tests if elements appear quickly.
Not recommended mixing with Explicit Waits.

```
driver.manage().timeouts().implicitlyWait(java.time.Duration.ofSeconds(10));
```

Sets the wait time to 10 seconds.

Disable Implicit Wait:

```
driver.manage().timeouts().implicitlyWait(java.time.Duration.ofSeconds(0));
```

Implicit waits poll the DOM at regular intervals until the element is found or the timeout expires.

Explicit Wait (WebDriverWait)

Waits for a specific *condition* to occur before proceeding.
Applied to specific elements/actions, not globally.
More flexible and powerful for complex scenarios.

```
WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(10));
WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("myElement")));
```

Waits up to 10 seconds for element with ID 'myElement' to be visible.

Common `ExpectedConditions`:

- `alertIsPresent()`
- `elementToBeClickable(By locator)`
- `elementToBeSelected(By locator)`
- `frameToBeAvailableAndSwitchToIt(By locator)` or `(String frameName)`
- `invisibilityOfElementLocated(By locator)`
- `presenceOfElementLocated(By locator)`
- `textToBePresentInElement(WebElement element, String text)`
- `titleIs(String title)`
- `urlContains(String fraction)`
- `visibilityOf(WebElement element)`
- `visibilityOfAllElementsLocatedBy(By locator)`

Waiting for multiple elements:

```
WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(15));
List<WebElement> elements = wait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(By.cssSelector(".item-list li")));
```

Ignoring exceptions during wait:

```
WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(10));
wait.ignoring(NoSuchElementException.class);
WebElement element = wait.until(ExpectedConditions.presenceOfElementLocated(By.id("possiblyAbsentElement")));
```

Fluent Wait

Defines maximum wait time, polling interval, and exceptions to ignore.
Most customizable wait.

```
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)

.withTimeout(java.time.Duration.ofSeconds(30))

.pollingEvery(java.time.Duration.ofSeconds(5))

.ignoring(NoSuchElementException.class);

WebElement element = wait.until(new java.util.function.Function<WebDriver, WebElement>() {
  public WebElement apply(WebDriver driver) {
    return driver.findElement(By.id("myElement"));
  }
});
```

Waits up to 30s, checking every 5s, ignores `NoSuchElementException`.

Fluent Wait is useful when you need different polling intervals or specific ignored exceptions for different waiting conditions.

Handling Specific Elements

Alerts (JavaScript Popups)

Switch to Alert	`Alert alert = driver.switchTo().alert();`
Accept Alert (OK)	`alert.accept();`
Dismiss Alert (Cancel)	`alert.dismiss();`
Get Alert Text	`String alertText = alert.getText();`
Send Keys to Prompt Alert	`alert.sendKeys("input text");`
Wait for Alert (using Explicit Wait)	`WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(10)); Alert alert = wait.until(ExpectedConditions.alertIsPresent()); alert.accept();`

Frames (iframes)

Switch by Index	`driver.switchTo().frame(0); // Switch to the first frame`
Switch by Name or ID	`driver.switchTo().frame("frameNameOrId");`
Switch by WebElement	`WebElement frameElement = driver.findElement(By.cssSelector("iframe[title='My Frame']")); driver.switchTo().frame(frameElement);`
Switch back to Parent Frame	`driver.switchTo().parentFrame();`
Switch back to Default Content (top level)	`driver.switchTo().defaultContent();`
Wait for Frame (using Explicit Wait)	`WebDriverWait wait = new WebDriverWait(driver, java.time.Duration.ofSeconds(10)); wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("myFrameId"))); // Now you are inside the frame // ... interact with elements ... driver.switchTo().defaultContent(); // Switch back when done`

Windows & Tabs

Get Current Window Handle	`String currentWindowHandle = driver.getWindowHandle();`
Get All Window Handles	`Set<String> allWindowHandles = driver.getWindowHandles();`
Switch to Window/Tab	`for (String handle : driver.getWindowHandles()) { if (!handle.equals(currentWindowHandle)) { driver.switchTo().window(handle); break; // Switch to the new window/tab } }`
Switch back to Original Window	`driver.switchTo().window(currentWindowHandle);`
Open New Tab (Selenium 4+)	`driver.switchTo().newWindow(WindowType.TAB); // New tab is automatically focused driver.get("http://newtab.com");`
Open New Window (Selenium 4+)	`driver.switchTo().newWindow(WindowType.WINDOW); // New window is automatically focused driver.get("http://newwindow.com");`
Close Specific Window/Tab	`// Assuming 'someHandle' is the handle of the window/tab to close driver.switchTo().window(someHandle).close(); // Remember to switch back if needed: // driver.switchTo().window(originalHandle);`

Advanced Interactions & Concepts

Mouse and Keyboard Actions

Use the `Actions` class for complex interactions.

```
Actions actions = new Actions(driver);
```

Perform a simple click using Actions:

```
WebElement element = driver.findElement(By.id("myButton"));
actions.click(element).perform();
```

Double Click:

```
WebElement element = driver.findElement(By.id("myButton"));
actions.doubleClick(element).perform();
```

Context Click (Right Click):

```
WebElement element = driver.findElement(By.id("myButton"));
actions.contextClick(element).perform();
```

Mouse Hover:

```
WebElement element = driver.findElement(By.id("myButton"));
actions.moveToElement(element).perform();
```

Drag and Drop (element to element):

```
WebElement source = driver.findElement(By.id("source"));
WebElement target = driver.findElement(By.id("target"));
actions.dragAndDrop(source, target).perform();
```

Drag and Drop (element by offset):

```
WebElement element = driver.findElement(By.id("source"));
actions.dragAndDropBy(element, 100, 50).perform(); // Move 100px
right, 50px down
```

Send Keys using Actions (useful for special keys):

```
WebElement input = driver.findElement(By.id("myInput"));
actions.sendKeys(input, "text").perform();
actions.sendKeys(Keys.ENTER).perform(); // Press ENTER key
```

Combined Actions (Build and Perform):

```
WebElement source = driver.findElement(By.id("source"));
WebElement target = driver.findElement(By.id("target"));
actions.moveToElement(source)
        .clickAndHold()
        .moveToElement(target)
        .release()
        .build()
        .perform();
```

Screenshots

Take a full page screenshot:

```
TakesScreenshot ts = (TakesScreenshot) driver;
File source = ts.getScreenshotAs(OutputType.FILE);
File destination = new File("./screenshots/screenshot.png");
FileUtils.copyFile(source, destination);
```

Requires Apache Commons IO dependency (`commons-io`).

```
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.15.1</version>
    <scope>test</scope>
</dependency>
```

Take screenshot of a specific element (Selenium 4+):

```
WebElement element = driver.findElement(By.id("myElement"));
File source = element.getScreenshotAs(OutputType.FILE);
File destination = new
File("./screenshots/element_screenshot.png");
FileUtils.copyFile(source, destination);
```

Saving screenshot to Base64 string:

```
TakesScreenshot ts = (TakesScreenshot) driver;
String base64Screenshot = ts.getScreenshotAs(OutputType.BASE64);
// Useful for embedding in reports
```

Page Object Model (POM)

A design pattern where web pages are represented as classes.
Each class contains WebElements and methods to interact with those elements.
Separates test logic from page interaction logic.
Improves code readability, maintainability, and reduces code duplication.

Basic Structure:

```
// src/main/java/pages/LoginPage.java
public class LoginPage {
    private WebDriver driver;

    // By locators (recommended)
    private By usernameInput = By.id("username");
    private By passwordInput = By.id("password");
    private By loginButton = By.id("loginButton");

    // Constructor
    public LoginPage(WebDriver driver) {
        this.driver = driver;
        // Optional: PageFactory.initElements(driver, this);
    }

    // Page methods
    public void enterUsername(String username) {
        driver.findElement(usernameInput).sendKeys(username);
    }

    public void enterPassword(String password) {
        driver.findElement(passwordInput).sendKeys(password);
    }

    public DashboardPage clickLoginButton() {
        driver.findElement(loginButton).click();
        return new DashboardPage(driver); // Return next page
object
    }

    public DashboardPage login(String username, String password)
{
        enterUsername(username);
        enterPassword(password);
        return clickLoginButton();
    }
}
```

Using PageFactory (Alternative element initialization):

```
// Inside LoginPage.java
import org.openqa.selenium.support.FindBy;
import org.openqa.selenium.support.PageFactory;

public class LoginPage {
    private WebDriver driver;

    @FindBy(id = "username")
    private WebElement usernameInput;

    @FindBy(id = "password")
    private WebElement passwordInput;

    @FindBy(id = "loginButton")
    private WebElement loginButton;

    public LoginPage(WebDriver driver) {
        this.driver = driver;
        PageFactory.initElements(driver, this); // Initialize
elements
    }

    public void enterUsername(String username) {
        usernameInput.sendKeys(username);
    }
    // ... other methods ...
}
```

Using POM in tests (`src/test/java/...Test.java`):

```
// Inside a TestNG/JUnit test method
WebDriver driver = new ChromeDriver();
LoginPage loginPage = new LoginPage(driver);

driver.get("http://myloginpage.com");
DashboardPage dashboardPage = loginPage.login("testuser",
"password123");

// Now interact with elements/methods on dashboardPage
String welcomeText = dashboardPage.getWelcomeMessage();
// Assert welcomeText...

driver.quit();
```