**Selenium New Features:**

Selenium 4.0

* announced in 2018
* 1st stable version - Oct 2021

**What's New in Selenium 4**

* Selenium is now W3C compliant
* Relative Locators
* Better Window/Tab Management
* Improved Selenium Grid
* Upgraded Selenium IDE
* New APIs for CDP (Chrome DevTools Protocol)
* Deprecation of Desired Capabilities
* Modifications in the Actions Class

**Selenium is now W3C compliant**

* JSON wire protocol was used to communicate between the Se Webdriver APIs and the browser native APIs
* With W3C compliance, the communication happens directly without any encoding and decoding required
* Any software following W3C standard protocol can be integrated with Selenium with no compatibility issues

All the major browsers e.g. Chrome, IE and Safari are already W3C compliant

**Relative Locators**

* Functions to locate nearby elements by specifying directions
* Existing locator strategy New relative locators

Id above

Name below

link Text toLeftOf

partial Link Text toRightOf

className near

tagName

Xpath

cssSelector

Example for relative locators:

Syntax: driver.findElement(RelativeLocator.with(By.id(“id”)).above(WebElement));

**Better window/tab management**

* work with multiple windows or tabs in the same session
* Can now open multiple windows/tabs without creating new driver object

Open new window and switch to the window

driver.switch To().newWindow(WindowType.WINDOW);

Open new tab and switch to the tab

driver.switch To().newWindow(WindowType.TAB);

**Improved Selenium Grid**

* Selenium Grid helps in Distributed Test Execution.
* Enables test execution on different combinations of browsers, OS, machines enables parallel execution

**Selenium Grid is now redesigned**

* Docker support
* Enables to spin up the containers. (NO need to set up VMs) enables to deploy the grid on Kubernetes for better scaling
* Easier Management - No need to set up and start hubs and nodes separately

**Improved Selenium Grid**

* 3 ways to run Se Grid

1. Standalone mode
2. Hub and Node
3. Distributed

**Standalone mode**

* The new Selenium Server Jar contains all the functionalities needed to run a grid java -jar selenium-server-4.0.0.jar standalone
* The Grid automatically identifies that the Webdriver for Chrome and Firefox are present on the system

**Hub and Node**

The classical way of using the Grid for Selenium test automation that consists of two major components - Hub and Nodes

* start hub : java -jar selenium-server-4.0.0.jar hub
* register node : java -jar selenium-server-4.0.0.jar node --detect-drivers

**Distributed**

Grid 4 can be started in a fully distributed manner, with each piece running in its own process

Selenium Grid 4 consists of FOUR processes

1. Router
2. Distributor
3. Node
4. Session

**Upgraded Selenium IDE**

Record & Playback tool

available as an add-on - Firefox, Chrome, MS Edge

* Improved GUI
* SIDE runner - Se IDE runner for CMD execution, grid and node projects
* Better element locator strategy
* More stable and reliable

**New APIs for CDP (Chrome DevTools Protocol)**

Chrome DevTools - set of tools built directly into Chromium-based browsers like Chrome, Opera, and Microsoft Edge to help developers debug and investigate websites

* Inspect Elements in the DOM
* Edit elements and CSS on the fly
* Check and monitor the site's performance
* Mock faster/slower networks speeds
* Mock geolocations of the user
* Execute and debug JavaScript
* View console logs

**New APIs for CDP (Chrome DevTools Protocol)**

* Se 4 comes with native support for Chrome DevTools APIs
* capture and monitor network traffic
* simulate poor network conditions
* perform geolocation testing
* Change device mode to do responsive design testing

new ChromiumDriver class, which includes two methods to access Chrome DevTools:

* + - getDev Tools()
    - executeCdpCommand()

**Deprecation of Desired Capabilities**

* Desired Capabilities were primarily used in the test scripts to define the test environment (browser name, version, operating system) for execution on the Selenium Grid
* capabilities objects are replaced with Options
  + - Firefox - FirefoxOptions
    - Chrome - ChromeOptions
    - Internet Explorer (IE) - InternetExplorerOptions
    - Microsoft Edge - EdgeOptions
    - Safari - SafariOptions

Example:

ChromeOptions options = new ChromeOptions();

options.setAccept InsecureCerts (true);

options.setCapability("build", "Testing Chrome Options [Selenium 4]");

options.setCapability ("name", "Testing Chrome Options [Selenium 4]");

options.setCapability ("platformName", "Windows 10");

options.setCapability("browserName", "Chrome");

options.setCapability("browserVersion", "latest");

**Modifications in the Actions Class:**

Actions class in Selenium is used to simulate input actions from mouse and keyboard on specific web elements (e.g. Left click, Right click, Double click, etc)

**Waits and Timeout:**

The parameters received in Timeout have switched from expecting (long time, TimeUnit unit) to expect (Duration duration).

**Before**:

**Frequency : 250 m/s**

driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

driver.manage().timeouts().setScriptTimeout(2, TimeUnit.MINUTES);

driver.manage().timeouts().pageLoadTimeout(10, TimeUnit.SECONDS);

**After:**

**Frequency : 500 m/s**

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

driver.manage().timeouts().scriptTimeout(Duration.ofMinutes(2));

driver.manage().timeouts().pageLoadTimeout(Duration.ofSeconds(10));

* Waits are also expecting different parameters now. WebDriverWait is now expecting a Duration instead of a long for timeout in seconds and milliseconds. The withTimeout and pollingEvery utility methods from FluentWait have switched from expecting (long time, TimeUnit unit) to expect (Duration duration).

**Before:**

* new WebDriverWait(driver, 3)

.until(ExpectedConditions.elementToBeClickable(By.cssSelector("#id")));

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

.withTimeout(30, TimeUnit.SECONDS)

.pollingEvery(5, TimeUnit.SECONDS)

.ignoring(NoSuchElementException.class);

**After:**

* new WebDriverWait(driver, Duration.ofSeconds(3)) .until(ExpectedConditions.elementToBeClickable(By.cssSelector("#id")));
* Wait<WebDriver> wait = new FluentWait<WebDriver>(driver) .withTimeout(Duration.ofSeconds(30))

.pollingEvery(Duration.ofSeconds(5))

.ignoring(NoSuchElementException.class);

**WebElement Screen shot:**

**getScreenShotAs**()

This method used to get screen shot of a WebElement.

Parameter type : OutputType

**Syntax**:

File source = chrome.findElement(By.*name*("q")).getScreenshotAs(OutputType.***FILE***);

File dest = **new** File("D:\\xxxx\\test.png");

FileUtils.*copyFile*(source, dest);