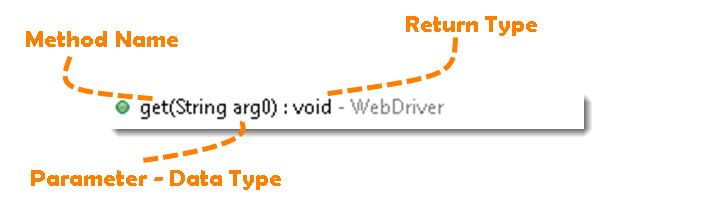
Now the next question is, How to access the methods of WebDriver? To check what all we have in WebDriver, create a driver object from WebDriver and press ***dot key.*** This will list down all the methods of WebDriver.



***Note:***Methodsfollowed by***Object***keyword are the generic methods gets from Object Class in Java. You will find these method for every object of java language.

* The suggestions marked in***Blue Color***are Nested Classes under WebDriver and will be covered in detail separately in the following chapters.
* The suggestions marked in***Green Color***are also Interfaces like WebDriver and will be covered in detail separately in the following chapters.
* The suggestions marked in***Violet Color***are similar methods like***Orange***but will be covered in detail separately in the following chapters.

Let’s just start discussing the ***Orange colored*** methods of ***Selenium WebDriver***but before that try to understand the syntax of the suggestions display by Eclipse for WebDriver.



**Method:** A Java method is a collection of statements that are grouped together to perform an operation.

* ***Method Name:***To access any method of any class, we need to create an object of class and then all the public methods will appear for the object.
* ***Parameter:***It is an argument which is passed to a method as a parameter to perform some operation. Every argument must passed with the same data type. For e.g.***get(String arg0) : void.***This is asking for a***String type***argument.
* ***Return Type:***Method can returns a value or returning nothing (void). If the***void***is mentioned after the method, it means the method is returning no value. And if it is returning any value, then it must display the type of the value for e.g.***getTitle() : String***.

Now it would be very easy to understand the WebDriver commands in the below chapter. The very first thing you like to do with Selenium is to ***Opening*** a new browser, ***Perform*** few tasks and***Closing*** the browser. Below are the numbers of commands you can apply on the Selenium opened browser.

## Get Command

***get(String arg0) : void*** - This method ***Load***a new web page in the current browser window. Accepts String as a parameter and returns nothing.

**Command**- ***driver.get(appUrl);***

Where **appUrl**is the website address to load. It is best to use a fully qualified URL.



|  |  |
| --- | --- |
| 1  2  3  4  5  6 | driver.get("http://www.google.com");    //Or can be written as    String URL = "http://www.DemoQA.com";  driver.get(URL); |

## Get Title Command

***getTitle() : String*** – This method fetches the ***Title*** of the current page. Accepts nothing as a parameter and returns a String value.

***Command – driver.getTitle();***

As the return type is String value, the output must be stored in String object/variable.



|  |  |
| --- | --- |
| 1  2  3  4  5 | driver.getTitle();    //Or can be used as    String Title = driver.getTitle(); |

## Get Current URL Command

***getCurrentUrl() : String*** – This method fetches the string representing the ***Current URL*** which is opened in the browser. Accepts nothing as a parameter and returns a String value.

***Command – driver.getCurrentTitle();***

As the return type is String value, the output must be stored in String object/variable.



|  |  |
| --- | --- |
| 1  2  3  4  5 | driver.getCurrentUrl();    //Or can be written as    String CurrentUrl = driver.getCurrentUrl(); |

## Get Page Source Command

***getPageSource() : String*** – This method returns the ***Source Code***of the page. Accepts nothing as a parameter and returns a String value.

***Command – driver.getPageSource();***

As the return type is String value, the output must be stored in String object/variable.



|  |  |
| --- | --- |
| 1  2  3  4 | driver.getPageSource();    //Or can be written as  String PageSource = driver.getPageSource(); |

## Close Command

***close() : void*** – This method **Close** only the current window the WebDriver is currently controlling. Accepts nothing as a parameter and returns nothing.

***Command – driver.close();***

Quit the browser if it’s the last window currently open.



|  |  |
| --- | --- |
| 1 | driver.close(); |

## Quit Command

***quit() : void*** – This method **Closes** all windows opened by the WebDriver. Accepts nothing as a parameter and returns nothing.

***Command – driver.quit();***

Close every associated window.



|  |  |
| --- | --- |
| 1 | driver.quit(); |

### Practice Exercise – 1

1. Launch a new Firefox browser.
2. Open Store.DemoQA.com
3. Get Page Title name and Title length
4. Print Page Title and Title length on the Eclipse Console.
5. Get Page URL and verify if the it is a correct page opened
6. Get Page Source (HTML Source code) and Page Source length
7. Print Page Length on Eclipse Console.
8. Close the Browser.

### Solution



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50 | package automationFramework;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  public class WebDriverCommands {    public static void main(String[] args) {  // Create a new instance of the FireFox driver  WebDriver driver = new FirefoxDriver();    // Storing the Application Url in the String variable  String url = "http://www.store.demoqa.com";    //Launch the ToolsQA WebSite  driver.get(url);    // Storing Title name in the String variable  String title = driver.getTitle();    // Storing Title length in the Int variable  int titleLength = driver.getTitle().length();    // Printing Title & Title length in the Console window  System.out.println("Title of the page is : " + title);  System.out.println("Length of the title is : "+ titleLength);    // Storing URL in String variable  String actualUrl = driver.getCurrentUrl();    if (actualUrl.equals(url)){  System.out.println("Verification Successful - The correct Url is opened.");  }else{  System.out.println("Verification Failed - An incorrect Url is opened.");  //In case of Fail, you like to print the actual and expected URL for the record purpose  System.out.println("Actual URL is : " + actualUrl);  System.out.println("Expected URL is : " + url);  }    // Storing Page Source in String variable  String pageSource = driver.getPageSource();    // Storing Page Source length in Int variable  int pageSourceLength = pageSource.length();    // Printing length of the Page Source on console  System.out.println("Total length of the Pgae Source is : " + pageSourceLength);    //Closing browser  driver.close();  }  } |

***Output***

Title of the page is : ONLINE STORE | Toolsqa Dummy Test site  
Length of the title is : 38  
Verification Failed – An incorrect Url is opened.  
Actual URL is : http://store.demoqa.com/  
Expected URL is : http://www.store.demoqa.com  
Total length of the Pgae Source is : 35646

### Practice Exercise – 2

1. Launch a new Firefox browser.
2. Open http://demoqa.com/frames-and-windows/
3. Use this statement to click on a New Window button “driver.findElement(By.xpath(“.//\*[@id=’tabs-1′]/div/p/a”)).click();”
4. Close the browser using close() command

You will notice that only one window will close. Next time use quit() command instead of close(). At that time selenium will close both the windows.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | package automationFramework;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;    public class WebDriverCommands\_2 {  public static void main(String[] args) {  WebDriver driver = new FirefoxDriver();  driver.get("http://demoqa.com/frames-and-windows/");  driver.findElement(By.xpath(".//\*[@id='tabs-1']/div/p/a")).click();  driver.close();  }  } |

**IMPLICIT VS EXPLICIT WAIT**

Introduce wait :

WebDriverWait wait = new WebDriverWait(webDriver, timeoutInSeconds);

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id<locator>));

or

wait.until(ExpectedConditions.elementToBeClickable(By.id<locator>));

With the explicit wait you have to insert a WebDriverWait.until(condition) before each findElement(). With the implicit wait it’s done for you, globally, after you make the driver timeouts setting. In general I like implicit wait, because the explicit WebDriverWait calls turn into clutter. I can’t think of a good reason why implicit wait is not the default.

1. **Explicit wait** WebDriverWait.until(*condition-that-finds-the-element*)
2. **Implicit wait** driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);

Here’s what the explicit wait looks like:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | public void testExplicitWait(WebDriver driver) {     // Assume we're already on a fully loaded webpage.     driver.findElement(By.id("myForm")).submit();     // Now the next page starts loading.       // Here's the explicit wait.     WebDriverWait wdw = new WebDriverWait(driver, 10);     ExpectedCondition<Boolean> condition = new ExpectedCondition<Boolean>() {        @Override        public Boolean apply(WebDriver d) {           WebElement result = d.findElement(By.className("myResult"));           return "The Next Page".equals(result.getText());           // Returns true as soon as an element of class 'myResult' is found           // where the element's text value is "The Next Page".       }     };     wdw.until(condition); // Won't get past here til timeout or element is found       // It is safe to operate on the element now.     WebElement result = driver.findElement(By.className("myResult"));     Assert.assertEquals("The Next Page", result.getText());  } |

Compare that to the implicit wait, which is much more concise:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public void testImplicitWait(WebDriver driver) {     // This setup would be done once per driver execution.     driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);       driver.findElement(By.id("myForm")).submit();     // Now the next page starts loading.       // An attempt to find the element implicitly waits til it is ready.     WebElement result = driver.findElement(By.className("myResult"));     Assert.assertEquals("The Next Page", result.getText());  } |

implicitlyWait only affects findElement, it does not affect other WebDriver methods likedriver.getCurrentUrl() or driver.manage().deleteAllCookies(). So you can very easily and accidentally get back the old url and/or delete cookies on the old subdomain, not the new url/subdomain you were expecting… unless you call findElement first to force an implicit wait:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19 | private void forceImplicitWait(WebDriver driver) {     driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);       driver.findElement(By.id("myForm")).submit();     // Now the next page starts loading ... but it's not done yet.       printCurrentUrl(driver); // Timing race!  Might print "/oldPath".       // An attempt to find an element on the next page forces an implicit wait     // til it is ready.  At that point the driver will have the new URL.     driver.findElement(By.className("myResult"));       printCurrentUrl(driver); // Safely prints "/newPath".  }    private void printCurrentUrl(WebDriver driver) {     URL currentUrl = null;     try {        currentUrl = new URL(driver.getCurrentUrl());     }     catch (MalformedURLException e) {        System.out.println("malformedUrl");     }     System.out.println(currentUrl.getPath()); |

Implicit Wait

Selenium WebDriver has borrowed the idea of **implicit waits** from **Watir**. This means that we can tell Selenium that we would like it to wait for a certain amount of time before throwing an **exception** that it cannot find the element on the page. We should note that implicit waits will be in place for the entire time the browser is open. This means that any search for elements on the page could take the time the implicit wait is set for.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | WebDriver driver = new FirefoxDriver();    driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);    driver.get("http://url\_that\_delays\_loading");    WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement")); |

## Fluent Wait

Each **FluentWait** instance defines the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Furthermore, the user may configure the wait to ignore specific types of exceptions whilst waiting, such as **NoSuchElementExceptions** when searching for an element on the page.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | // Waiting 30 seconds for an element to be present on the page, checking      // for its presence once every 5 seconds.      Wait wait = new FluentWait(driver)        .withTimeout(30, SECONDS)        .pollingEvery(5, SECONDS)        .ignoring(NoSuchElementException.class);      WebElement foo = wait.until(new Function() {        public WebElement apply(WebDriver driver) {        return driver.findElement(By.id("foo"));        }       }); |

## Explicit Wait

It is more extendible in the means that you can set it up to wait for any condition you might like. Usually, you can use some of the prebuilt **ExpectedConditions** to wait for elements to become clickable, visible, invisible, etc.



|  |  |
| --- | --- |
| 1  2  3 | WebDriverWait wait = new WebDriverWait(driver, 10);    WebElement element = wait.until(ExpectedConditions.elementToBeClickable(By.id("someid"))); |

# Difference Between Implicit, Explicit and Fluent Wait

**Implicit Wait:** During Implicit wait if the Web Driver cannot find it immediately because of its availability, the WebDriver will wait for mentioned time and it will not try to find the element again during the specified time period. Once the specified time is over, it will try to search the element once again the last time before throwing exception. The default setting is zero. Once we set a time, the Web Driver waits for the period of the WebDriver object instance.

**Explicit Wait:** There can be instance when a particular element takes more than a minute to load. In that case you definitely not like to set a huge time to Implicit wait, as if you do this your browser will going to wait for the same time for every element.

To avoid that situation you can simply put a separate time on the required element only. By following this your browser implicit wait time would be short for every element and it would be large for specific element.

**Fluent Wait:**Let’s say you have an element which sometime appears in just 1 second and some time it takes minutes to appear. In that case it is better to use fluent wait, as this will try to find element again and again until it find it or until the final timer runs out.

**Solutions:**We always get confuse when it comes to using Wait commands, to better understand it we need to remember that there is a difference between several scenarios:

An element not being present at all in the DOM.

An element being present in the DOM but not visible.

An element being present in the DOM but not enabled. (i.e. clickable)

There are pages which get displayed with the **JavaScript**, the elements are already present in the browser **DOM**, but are not visible. The implicit wait only waits for an element to appear in the DOM, so it returns immediately, but when you try to interact with the element you get a**NoSuchElementException**. You could test this hypothesis by writing a helper method that explicit wait for an element to be visible or clickable.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public WebElement getWhenVisible(By locator, int timeout) {    WebElement element = null;    WebDriverWait wait = new WebDriverWait(driver, timeout);    element = wait.until(ExpectedConditions.visibilityOfElementLocated(locator));    return element;    } |



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | public void clickWhenReady(By locator, int timeout) {    WebElement element = null;    WebDriverWait wait = new WebDriverWait(driver, timeout);    element = wait.until(ExpectedConditions.elementToBeClickable(locator));    element.click();    } |