## Untrusted SSL Certificate

***Internet Explorer*** is the product of ***Microsoft*** and IE is much worried about security and IE is known as the most secured browser. At times using IE Browser with Selenium gives ***SLL Certificate*** pop up.

There are two ways to resolve the SLL Certificate issue.

### *****Solution One*****

Add the below script just under the code to open the application:

driver.navigate().to(“javascript:document.getElementById(‘overridelink’).click()”);

Complete code will look like this:



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | // Change the package name accordingly, as per your project  package selenium;  import org.openqa.selenium.ie.InternetExplorerDriver;  public class IEExplorerTest {    public static void main(String[] args) {  //Path to the folder where you have extracted the IEDriverServer executable  String service = "D:\\ToolsQA\\trunk\\Library\\drivers\\IEDriverServer.exe";  System.setProperty("webdriver.ie.driver", service);    InternetExplorerDriver  driver = new InternetExplorerDriver();  driver.get("URL for which certificate error is coming");  driver.navigate().to("javascript:document.getElementById('overridelink').click()");  }  } |

### Solution Two

Another way of avoiding this error is to use DesiredCapability settings of the browser.



|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | // Change the package name accordingly, as per your project  package selenium;  import org.openqa.selenium.ie.InternetExplorerDriver;  import org.openqa.selenium.remote.CapabilityType;  import org.openqa.selenium.remote.DesiredCapabilities;  public class IEExplorerTest {    public static void main(String[] args) {  // Path to the folder where you have extracted the IEDriverServer executable  String service = "D:\\ToolsQA\\trunk\\Library\\drivers\\IEDriverServer.exe";  System.setProperty("webdriver.ie.driver", service);    // Create the DesiredCapability object of InternetExplorer  DesiredCapabilities capabilities = DesiredCapabilities.internetExplorer();    // Settings to Accept the SSL Certificate in the Capability object  capabilities.setCapability(CapabilityType.ACCEPT\_SSL\_CERTS, true);    InternetExplorerDriver driver = new InternetExplorerDriver(capabilities);  driver.get("URL for which certificate error is coming");    }  } |

After putting above code, run your script, this time no SSL Certificate Error would appear on screen and script would run fine.

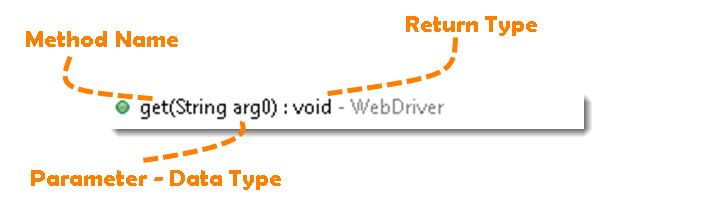
Now the next question is, How to access the methods of WebDriver? To check what all we have in WebDriver, create a driver object from WebDriver and press ***dot key.*** This will list down all the methods of WebDriver.



***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

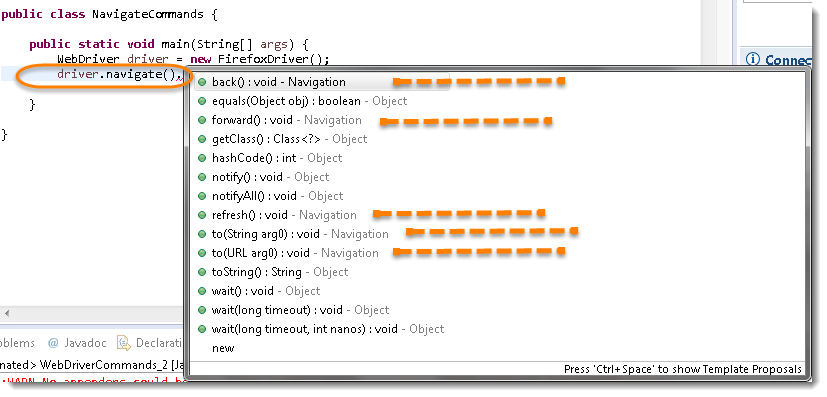
1. Launch a new Firefox browser.
2. Open http://demoqa.com/frames-and-windows/
3. Use this statement to click on a New Window button “driver.findElement(By.xpath(“.//\*[@id=’tabs-1′]/div/p/a”)).click();”
4. Close the browser using close() command

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();  }  } |

fter successfully running our first test case on Firefox Browser now we are stepping towards grasping the essential ***Browser Navigation Commands*** in Selenium. Thus we are going to discuss about various navigation commands that we would be using in our day to day automation testing. The navigate interface exposes the ability to move backwards and forwards in the browser’s history.

To access the navigation’s method, just type ***driver.navigate().***. The intellisence feature of eclipse will automatically display all the public methods of **Navigate Interface**shown in the below image.  


***Note:***Only methods which are followed by***Navigation***keyword are belongs to navigate. Rest followed by***Object***keyword are the generic methods gets from Object Class in Java. You will find these method for every object of java language.

## Navigate To Command

***to(String arg0) : void*** – This method ***Loads***a new web page in the current browser window. It accepts a String parameter and returns nothing.

**Command** - ***driver.navigate().to(appUrl);***

It does exactly the same thing as the ***driver.get(appUrl)*** method. Where **appUrl**is the website address to load. It is best to use a fully qualified URL.



|  |  |
| --- | --- |
| 1 | driver.navigate().to("http://www.DemoQA.com"); |

## Forward Command

***forward() : void*** – This method does the same operation as clicking on the ***Forward Button*** of any browser. It neither accepts nor returns anything.

**Command** - ***driver.navigate().forward();***

Takes you forward by one page on the browser’s history.



|  |  |
| --- | --- |
| 1 | driver.navigate().forward(); |

## Back Command

***back() : void*** – This method does the same operation as clicking on the ***Back Button*** of any browser. It neither accepts nor returns anything.

**Command** - ***driver.navigate().back();***

Takes youback by one page on the browser’s history.



|  |  |
| --- | --- |
| 1 | driver.navigate().back(); |

## Refresh Command

***refresh() : void*** – This method ***Refresh*** the current page. It neither accepts nor returns anything.

**Command** - ***driver.navigate().refresh();***

Perform the same function as pressing F5 in the browser.



|  |  |
| --- | --- |
| 1 | driver.navigate().refresh(); |

### Practice Exercise

1. Launch new Browser
2. Open DemoQA.com website
3. Click on Registration link using “driver.findElement(By.*xpath*(“.//\*[@id=’menu-item-374′]/a”)).click();“
4. Come back to Home page (Use ‘Back’ command)
5. Again go back to Registration page (This time use ‘Forward’ command)
6. Again come back to Home page (This time use ‘To’ command)
7. Refresh the Browser (Use ‘Refresh’ command)
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 | package automationFramework;  import org.openqa.selenium.By;  import org.openqa.selenium.WebDriver;  import org.openqa.selenium.firefox.FirefoxDriver;  public class NavigateCommands {  public static void main(String[] args) {  // Create a new instance of the FireFox driver  WebDriver driver = new FirefoxDriver();    // Open ToolsQA web site  String appUrl = "http://www.DemoQA.com";  driver.get(appUrl);    // Click on Registration link  driver.findElement(By.xpath(".//\*[@id='menu-item-374']/a")).click();    // Go back to Home Page  driver.navigate().back();    // Go forward to Registration page  driver.navigate().forward();    // Go back to Home page  driver.navigate().to(appUrl);    // Refresh browser  driver.navigate().refresh();    // Close browser  driver.close();  }  }  So far in our Selenium Learning journey we have done ***WebDriver Commands*** and ***Navigation Commands***. Soon we will be identifying the different ***WebElement*** on webpages and performing various actions on it. This chapter is all about **Selenium WebDriver WebElement Commands**. But before moving on to finding different WebElements, it better to cover that what all operations we can perform on a WebElement. In this chapter we will learn ***What is WebElement*** and the ***List of Actions*** can be performed on various WebElements.   What is WebElement? WebElement represents an***HTML element***. HTML documents are made up by HTML elements. HTML elements are written with a ***start*** tag, with an ***end***tag, with the **content** in between: ***<tagname> content </tagname>***  The HTML **element** is everything from the start tag to the end tag: ***<p> My first HTML paragraph. </p>***  HTML elements can be nested (elements can contain elements). All HTML documents consist of nested HTML elements.     |  |  | | --- | --- | | 1  2  3  4  5  6 | <html>  <body>  <h1> My First Heading </h1>  <p> My first paragraph. </p>  </body>  </html> |    List of WebElement Commands/Actions All interesting operations to do with interacting with a page will be performed through this ***WebElement Interface.***  WebElementCommands_01  ***Note:***Methods followed by***Object***keyword are the generic methods gets from Object Class in Java. You will find these methods for every object of java language.  Before going through each and every action of WebElement, let’s just understand that how we get a WebElement object/element. As in the previous chapters we learned that every method of the **WebDriver** either returns something or return void(means return nothing). The same way ***findElement***command of ***WebDriver*** returns ***WebElement***. WebElementCommands_02  So, to get the WebElement object write the below statement:  ***WebElement element = driver.findElement(By.id(“UserName“));***  And now if you type ***element dot***, Eclipse’s intellisence will populate the complete list of actions just like the above image.    One more thing to notice that WebElement can be of any type, like it can be a***Text, Link, Radio Button, Drop Down, WebTable*** or any HTML element. But all the actions will always populate against the any element irrespective of whether the action is valid on the WebElement or not. For e.g. ***clear() command***, even if you have a link element still you get the option to chooseclear() command on it, which if you choose may result in some error or may not does anything.   Clear Command ***clear( ) : void*** - If this element is a text entry element, this will clear the value. This method accepts nothing as a parameter and returns nothing.  **Command**- ***element.clear();***  This method has no effect on other elements. Text entry elements are ***INPUT***and ***TEXTAREA*** elements.     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.id("UserName"));  element.clear();    //Or can be written as    driver.findElement(By.id("UserName")).clear(); |    SendKeys Command ***sendKeys(CharSequence… keysToSend ) : void*** - This simulate typing into an element, which may set its value. This method accepts CharSequence as a parameter and returns nothing.  **Command**- ***element.sendKeys(“text”);***  This method works fine with text entry elements like ***INPUT*** and ***TEXTAREA***elements.     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.id("UserName"));  element.sendKeys("ToolsQA");    //Or can be written as    driver.findElement(By.id("UserName")).sendKeys("ToolsQA"); |    Click Command ***click( ) : void*** - This simulates the clicking of any element. Accepts nothing as a parameter and returns nothing.  **Command**- ***element.click();***  Clicking is perhaps the most common way of interacting with web elements like text elements, links, radio boxes and many more**.**     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.linkText("ToolsQA"));  element.click();    //Or can be written as    driver.findElement(By.linkText("ToolsQA")).click(); |   ***Note***: Most of the time we click on the links and it causes a new page to load, this method will attempt to wait until the page has loaded properly before handing over the execution to next statement. But If click() causes a new page to be loaded via an event or is done by sending a native event for example through javascript, then the method will not wait for it to be loaded.  There are some preconditions for an element to be clicked. The element must be***Visible***and it must have a***Height and Width***greater than 0.   IsDisplayed Command ***isDisplayed( ) : boolean*** - This method determines if an element is currently being displayed or not. This accepts nothing as a parameter but returns boolean value(true/false).  **Command**- ***element.isDisplayed();***     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.id("UserName"));  boolean status = element.isDisplayed();    //Or can be written as    boolean staus = driver.findElement(By.id("UserName")).isDisplayed(); |   ***Note***: Do not confuse this method with element present on the page or not. This will return***true*** if the element is present on the page and throw a***NoSuchElementFound*** exception if the element is not present on the page. This refers the property of the element, sometimes the element is present on the page but the property of the element is set to ***hidden***, in that case this will return ***false***, as the element is present in the DOM but not visible to us.   IsEnabled Command ***isEnabled( ) : boolean*** - This determines if the element currently is ***Enabled or not***? This accepts nothing as a parameter but returns boolean value(true/false).  **Command**- ***element.isEnabled();***  This will generally return true for everything but I am sure you must have noticed many disabled input elements in the web pages.     |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14 | WebElement element = driver.findElement(By.id("UserName"));  boolean status = element.isEnabled();    //Or can be written as    boolean staus = driver.findElement(By.id("UserName")).isEnabled();    //Or can be used as  WebElement element = driver.findElement(By.id("userName"));  boolean status = element.isEnabled();  // Check that if the Text field is enabled, if yes enter value  if(status){      element.sendKeys("ToolsQA");  } |    IsSelected Command ***isSelected( ) : boolean*** - Determine whether or not this element is selected or not. This accepts nothing as a parameter but returns boolean value(true/false).  **Command**- ***element.isSelected();***  This operation only applies to input elements such as ***Checkboxes***, **Select Options** and ***Radio Buttons***. This returns ***True*** if the element is currentlyselected or checked, ***false*** otherwise.     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.id("Sex-Male"));  boolean status = element.isSelected();    //Or can be written as    boolean staus = driver.findElement(By.id("Sex-Male")).isSelected(); |   ***Note***: In the later chapters of [***Check Box & Radio Buttons***](http://toolsqa.com/selenium-webdriver/checkbox-radio-button-operations/) and [***Drop Down & Multiple Selects***](http://toolsqa.com/selenium-webdriver/dropdown-multiple-select-operations/)**,**we have covered many examples around it.   Submit Command ***submit( ) : void***- This method works well/better than the click() if the current element is a form, or an element within a form. This accepts nothing as a parameter and returns nothing.  **Command**- ***element.submit();***  If this causes the current page to change, then this method will wait until the new page is loaded.     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.id("SubmitButton"));  element.submit();    //Or can be written as    driver.findElement(By.id("SubmitButton")).submit(); |    GetText Command ***getText( ) : String***- This method will fetch the visible (i.e. not hidden by CSS) innerText of the element. This accepts nothing as a parameter but returns a String value.  **Command**- ***element.submit();***  This returns an innerText of the element, including sub-elements, without any leading or trailing whitespace.     |  |  | | --- | --- | | 1  2 | WebElement element = driver.findElement(By.xpath("anyLink"));  String linkText = element.getText(); |    getTagName Command ***getTagName( ) : String***- This method gets the tag name of this element. This accepts nothing as a parameter and returns a String value.  **Command**- ***element.getTagName();***  This does not return the value of the name attribute but return the tag for e.g. “***input***“ for the element***<input name="foo"/>***.     |  |  | | --- | --- | | 1  2  3  4  5  6 | WebElement element = driver.findElement(By.id("SubmitButton"));  String tagName = element.getTagName();    //Or can be written as    String tagName = driver.findElement(By.id("SubmitButton")).getTagName(); |    getCssValue Command ***getCssvalue( ) : String***- This method Fetch CSS property value of the give element. This accepts nothing as a parameter and returns a String value.  **Command**- ***element.getCssValue();***  Color values should be returned as rgba strings, so, for example if the “background-color” property is set as “green” in the HTML source, the returned value will be “rgba(0, 255, 0, 1)”.   getAttribute Command ***getAttribute(String Name) : String***- This method gets the value of the given attribute of the element. This accepts the String as a parameter and returns a String value.  **Command**- ***element.getAttribute();***  Attributes are Ids, Name, Class extra and using this method you can get the value of the attributes of any given element.     |  |  | | --- | --- | | 1  2 | WebElement element = driver.findElement(By.id("SubmitButton"));  String attValue = element.getAttribute("id"); //This will return "SubmitButton" |    getSize Command ***getSize( ) : Dimension***- This method fetch the width and height of the rendered element. This accepts nothing as a parameter but returns the Dimension object.  **Command**- ***element.getSize();***  This returns the size of the element on the page.     |  |  | | --- | --- | | 1  2  3 | WebElement element = driver.findElement(By.id("SubmitButton"));  Dimension dimensions = element.getSize();  System.out.println(“Height :” + dimensions.height + ”Width : "+ dimensions.width); |    getLocation Command ***getLocation( ) : Point***- This method locate the location of the element on the page. This accepts nothing as a parameter but returns the Point object.  **Command**- ***element.getLocation();***  This returns the ***Point object***, from which we can get X and Y coordinates of specific element.     |  |  | | --- | --- | | 1  2  3 | WebElement element = driver.findElement(By.id("SubmitButton"));  Point point = element.getLocation();  System.out.println("X cordinate : " + point.x + "Y cordinate: " + point.y); | |

**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();    } |