Handling Web Elements and Mouse Events

29TH MAY, 2023

expleo

Contents

Why do we need Selenium Waits?

Introduction

- Majority of modern application's front-end is built on JavaScript or Ajax, using frameworks such as React, Angular, or any other which takes a certain time for the web elements to load on the page, whenever that page is loaded or refreshed.
- Hence, in case you tend to locate an element in your script which is yet to load on the page, selenium will throw you 'Exception' message.
- Three different exceptions.

1. The Element we try to interact/find is not at all present in the DOM.
2. Element is present but not visible.
3. Element is present but not interactable.

An Example Scenario

- Below code snippet will help you showcase the same problem as you execute automation testing with Selenium.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class NoWaitImplemented {
 public static void main(String[] args) throws InterruptedException {
 System.setProperty("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");
 WebDriver driver=new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://omayo.blogspot.com/");
 driver.findElement(By.className("dropbtn")).click();
 driver.findElement(By.linkText("Facebook")).click();
 }
}
```

Why do we need selenium waits?

An Example Scenario

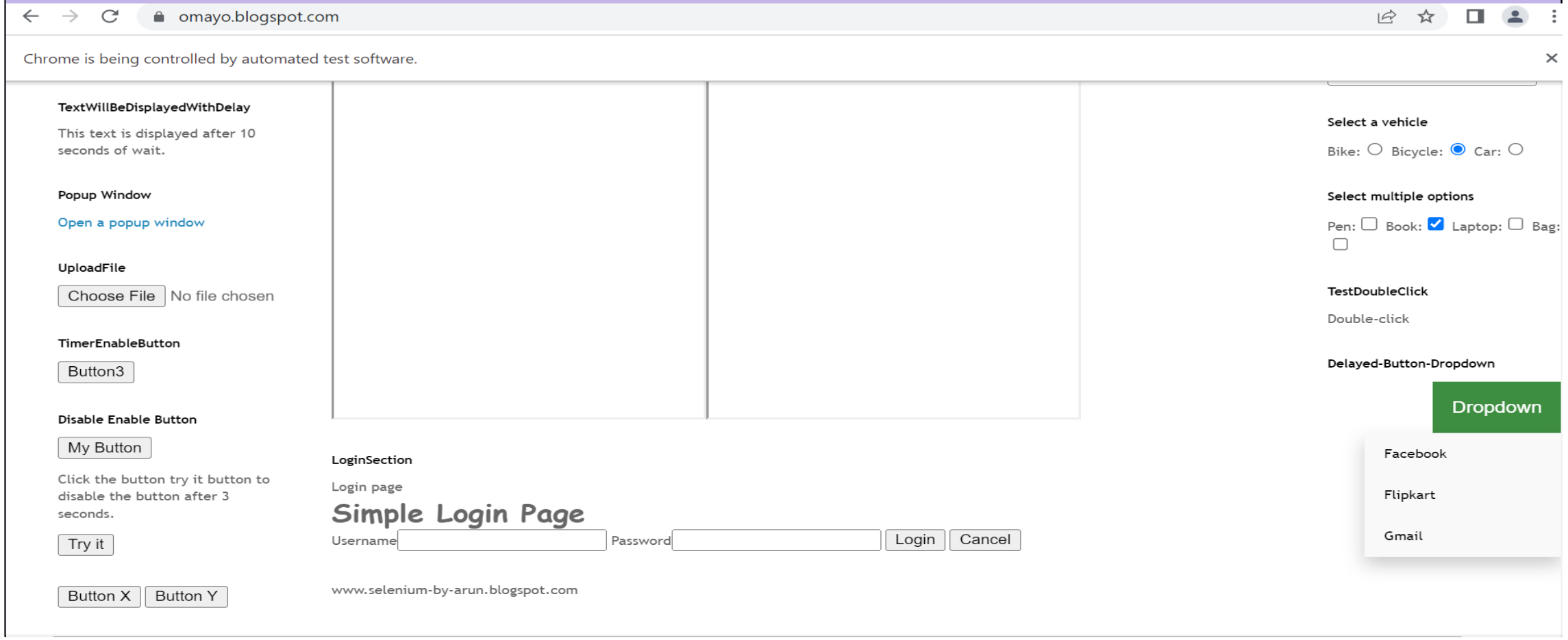
Output:

```
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Starting ChromeDriver 111.0.5563.64 (c710e93d5b63b7095afe8c2c17df34408078439d-refs/branch-heads/5563@{#995}) on port 22325
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
[1683007539.738][WARNING]: This version of ChromeDriver has not been tested with Chrome version 112.
May 02, 2023 11:35:40 AM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 112, so returning the closest version found: 111
Exception in thread "main" org.openqa.selenium.NoSuchElementException: no such element: Unable to locate element: {"method":"link text","selector":"Facebo
  (Session info: chrome=112.0.5615.138)
For documentation on this error, please visit: https://selenium.dev/exceptions/#no_such_element
Build info: version: '4.8.3', revision: 'b19b418e60'
System info: os.name: 'Windows 10', os.arch: 'amd64', os.version: '10.0', java.version: '17.0.5'
Driver info: org.openqa.selenium.chrome.ChromeDriver
Command: [07f81438d1477f3098d6f6c8bf50b9f9, findElement {using=link text, value=Facebook}]
Capabilities {acceptInsecureCerts: false, browserName: chrome, browserVersion: 112.0.5615.138, chrome: {chromedriverVersion: 111.0.5563.64 (c710e93d5b63..
```

Why do we need selenium waits?

An Example Scenario

Output:



expleo

Introduction

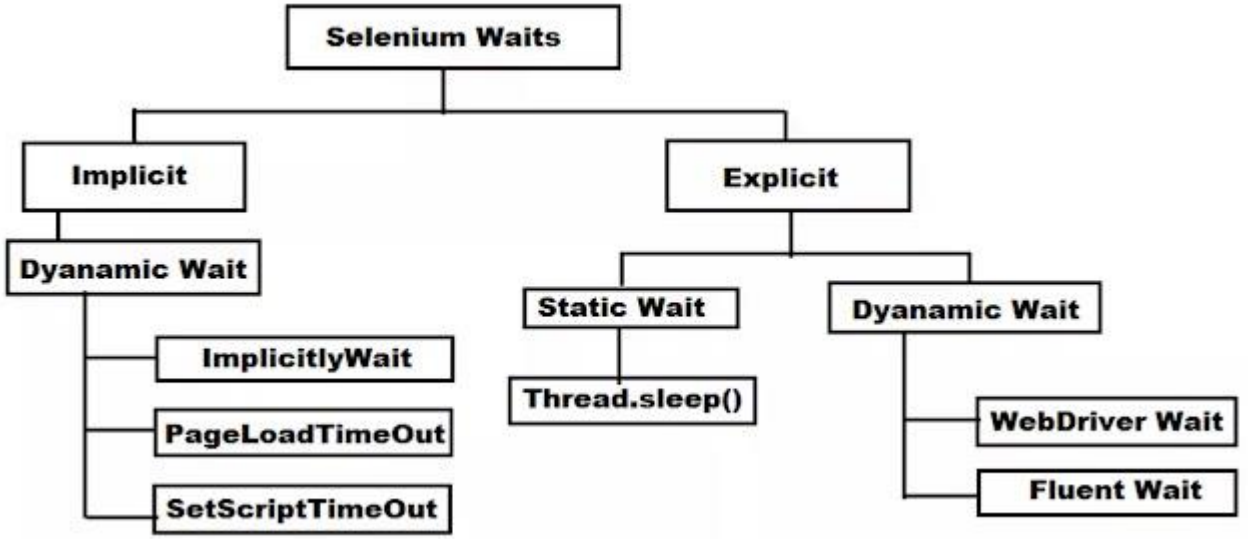
- Now, since the page hasn’t loaded yet, the script failed to find the ‘Facebook’ button. Resulting in throwing a ‘NoSuchElementException’.
- Selenium wait for a page to load helps in resolving this issue.
- There are different types of Selenium waits like Implicit wait and Explicit wait, that ensures the elements are loaded into the page before they are discovered by the Selenium script for further actions.

Types of Selenium Waits

Selenium Waits

- When performing automation testing with Selenium, we use the following types of waits as we generate our Selenium script:

- Thread.Sleep() method
- Implicit Wait
- Explicit Wait
- Fluent Wait



About Script Synchronization in selenium

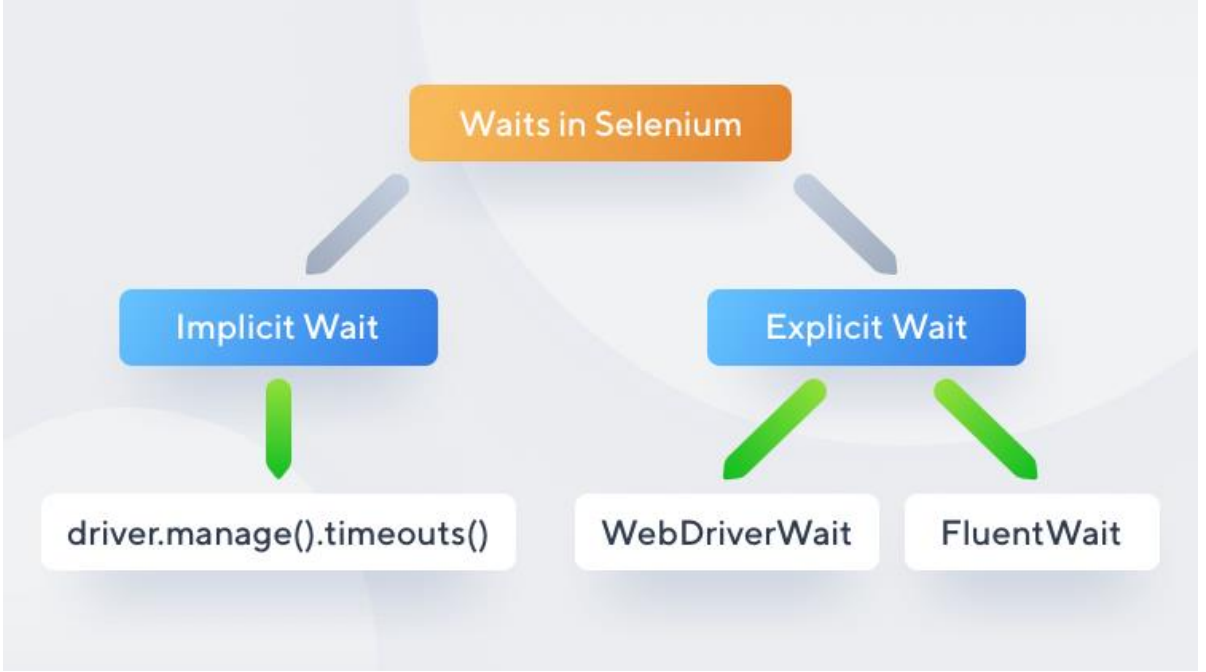
✓ Synchronize between script execution and application, we need to wait after performing appropriate actions.

Some of Waits used in Selenium are :

1)Implicit Wait

2)Explicit Wait

3)Fluent Wait



Thread.Sleep() and its disadvantages

Thread.Sleep()

- Sleep is a static method that belongs to the thread class.
- This method can be called using the reference of the class name i.e Thread.
- If you use Thread.sleep while performing automation testing with Selenium, then this method will stop the execution of the script for the specified duration of time, irrespective of whether the element is found or not on the web page.
- It accepts the time duration in milliseconds.
- The syntax for the same is:

```
Thread.sleep(3000);
```

An Example Scenario

- Below code snippet will help you showcase the same problem as you execute automation testing with Selenium.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class Threadsleep{
 public static void main(String[] args) throws InterruptedException {
 System.setProperty("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");
 WebDriver driver=new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://omayo.blogspot.com/");
 driver.findElement(By.className("dropbtn")).click();
 Thread.sleep(5000);
 driver.findElement(By.linkText("Facebook")).click();
 }
}
```

An Example Scenario

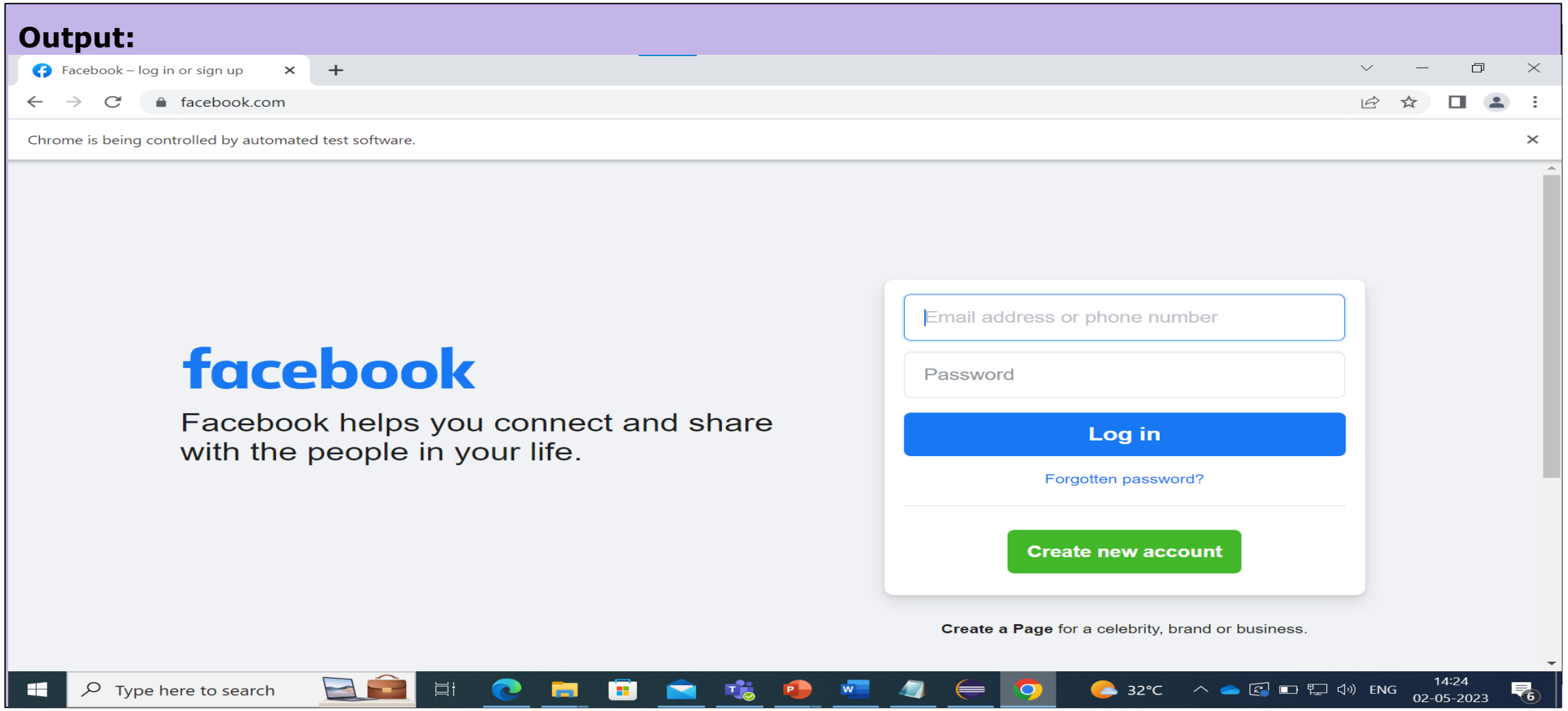
Output:

```
SLF4J: No SLF4J providers were found.
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See https://www.slf4j.org/codes.html#noProviders for further details.
Starting ChromeDriver 111.0.5563.64 (c710e93d5b63b7095afe8c2c17df34408078439d-refs/branch-heads/5563@{#995}) on port 41671
Only local connections are allowed.
Please see https://chromedriver.chromium.org/security-considerations for suggestions on keeping ChromeDriver safe.
ChromeDriver was started successfully.
[1683013379.210][WARNING]: This version of ChromeDriver has not been tested with Chrome version 112.
May 02, 2023 1:12:59 PM org.openqa.selenium.devtools.CdpVersionFinder findNearestMatch
WARNING: Unable to find an exact match for CDP version 112, so returning the closest version found: 111
```

Why do we need selenium waits?

An Example Scenario

Output:



expleo

Why Using Thread.Sleep() Is Not A Good Idea?

- Selenium Webdriver waits for the specified time, irrespective of the element is found or not.
- In case the element is found much before the specified duration, the script will still wait for the time duration to elapse, thereby increasing the execution time of the script.
- If the element to be present does not appear after a static time and keep changing, then you will never know an estimated time needed for the sleep function.
- Thread.sleep is intended only for the element it is written prior to. In case you have two to fours elements which need to wait for a certain duration to load, you need to specify Thread.sleep as many times in that case.

Implicit Wait

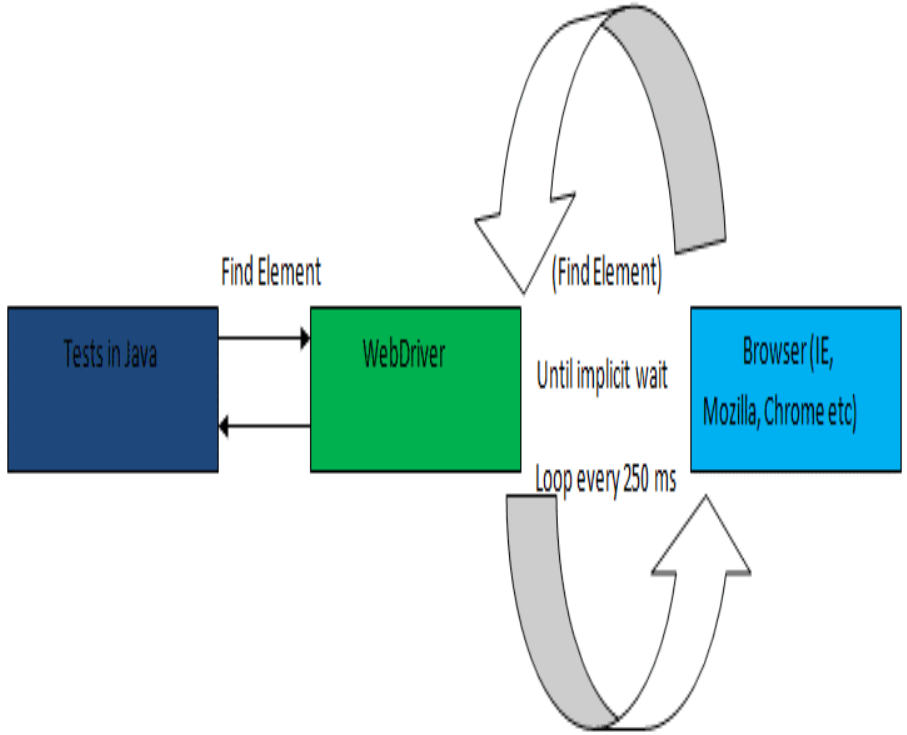
Implicit Wait

- Selenium has overcome the problems provided by Thread.sleep() and have come up with two Selenium waits for page load.
- One of which is Implicit wait which allows you to halt the WebDriver for a particular period of time until the WebDriver locates a desired element on the web page.
- The key point to note here is, unlike Thread.sleep(), it does not wait for the complete duration of time.
- This is why Implicit wait is also referred to as dynamic wait. If it does not find the element in the specified duration, it throws ElementNotVisibleException.

Implicit Wait

- Another interesting thing to note about Implicit wait is that it is applied globally, which makes it a better option than Thread.sleep().
- Meaning you only need to write it one time and it gets applicable for all of the web elements specified on a script throughout the WebDriver instance.
- To add implicit waits in test scripts, import the following package:

import java.util.concurrent.TimeUnit;



Implicit Wait

- Syntax:

```
driver.manage().timeouts().implicitlyWait(Time Interval to wait for,TimeUnit.SECONDS);

driver.manage().timeouts().implicitlyWait(TimeUnit(Time Interval));
```

- Example:

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

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

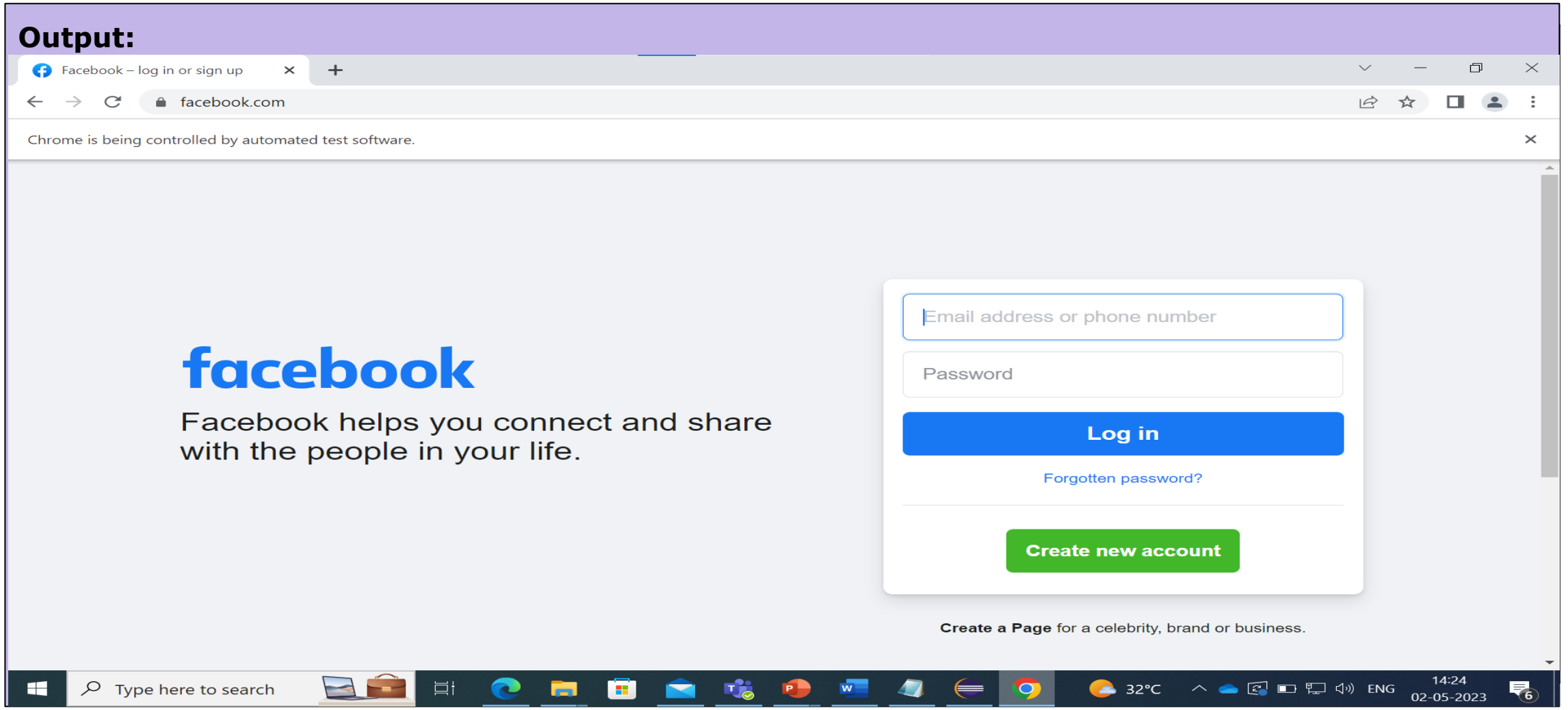
An Example Scenario

- Below code snippet will help you showcase the same problem as you execute automation testing with Selenium.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
public class Implicit {
 public static void main(String[] args) throws InterruptedException {
 System.setProperty("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");
 WebDriver driver=new ChromeDriver();
 driver.manage().window().maximize();
 driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));
 driver.get("https://omayo.blogspot.com/");
 driver.findElement(By.className("dropbtn")).click();
 driver.findElement(By.linkText("Facebook")).click();
 }
}
```

Example

Output:



Explicit Wait

Explicit Wait

- Now, here we are aware of the fact, that pages shall we loaded in a certain duration, but what if we do not know the element to be visible/clickable at loading time.
- As in the time of its appearance is dynamic and keeps on changing from time to time. In this case, Explicit wait will help you overcome this problem.
- The Explicit wait is another one of the dynamic Selenium waits.
- Explicit wait help to stop the execution of the script based on a certain condition for a specified amount of time.
- Once the time goes overboard, you will get the ElementNotVisibleException.
- In a scenario where you do not know the amount of time to wait for, this explicit wait comes in handy.

Explicit Wait

- Using conditions like elementToBeClickable() or textToBePresentInElement(), one can wait for the specified duration.
- One can use these predefined methods using the combination of classes WebDriverWait and ExpectedConditions.
- In order to use this case, import the below packages in your class:

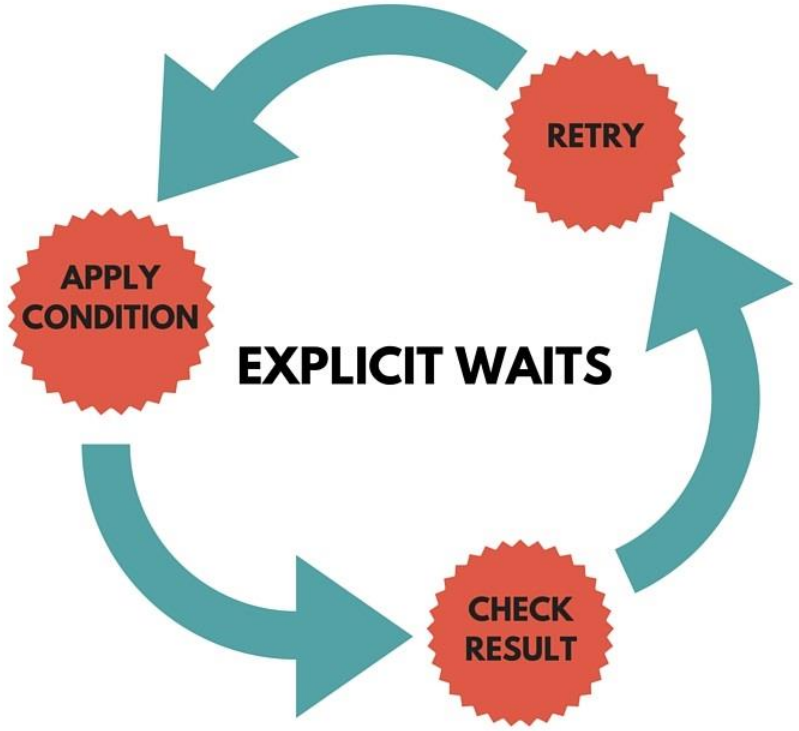
```
import org.openqa.selenium.support.ui.ExpectedConditions

import org.openqa.selenium.support.ui.WebDriverWait
```

Explicit Wait

- Then, Initialize A Wait Object using WebDriverWait Class.

```
WebDriverWait wait = new WebDriverWait(driver,30);
```

- Here, the reference variable is named <wait> for the <WebDriverWait> class.
- It is instantiated using the WebDriver instance.
- The maximum wait time must be set for the execution to layoff.
- Note that the wait time is measured in seconds.



Explicit Wait

- In order to use the predefined methods of the ExpectedCondition Class, we will use the wait reference variable as below:
- Types of Expected Conditions:
- Below are the few types of expected conditions commonly used as you perform automation testing with Selenium.
- visibilityOfElementLocated()- Verifies if the given element is present or not
- alertIsPresent()- Verifies if the alert is present or not.
- elementToBeClickable()- Verifies if the given element is present/clickable on the screen
- textToBePresentInElement()- Verifies the given element have the required text or not
- titleIs()- Verify the condition wait for a page that has a given title

Explicit Wait

Syntax:

- wait.until(ExpectedConditions.visibilityOfElementLocated(Reference of Element to be located using locator));

Example:

- wait.until(ExpectedConditions.visibilityOf(driver.findElement(By.xpath("//div[@class='Campaign__innerWrapper']/button"))));
- Selenium 4 Syntax
- WebDriverWait **wait** = new WebDriverWait(driver,Duration.ofSeconds(10));

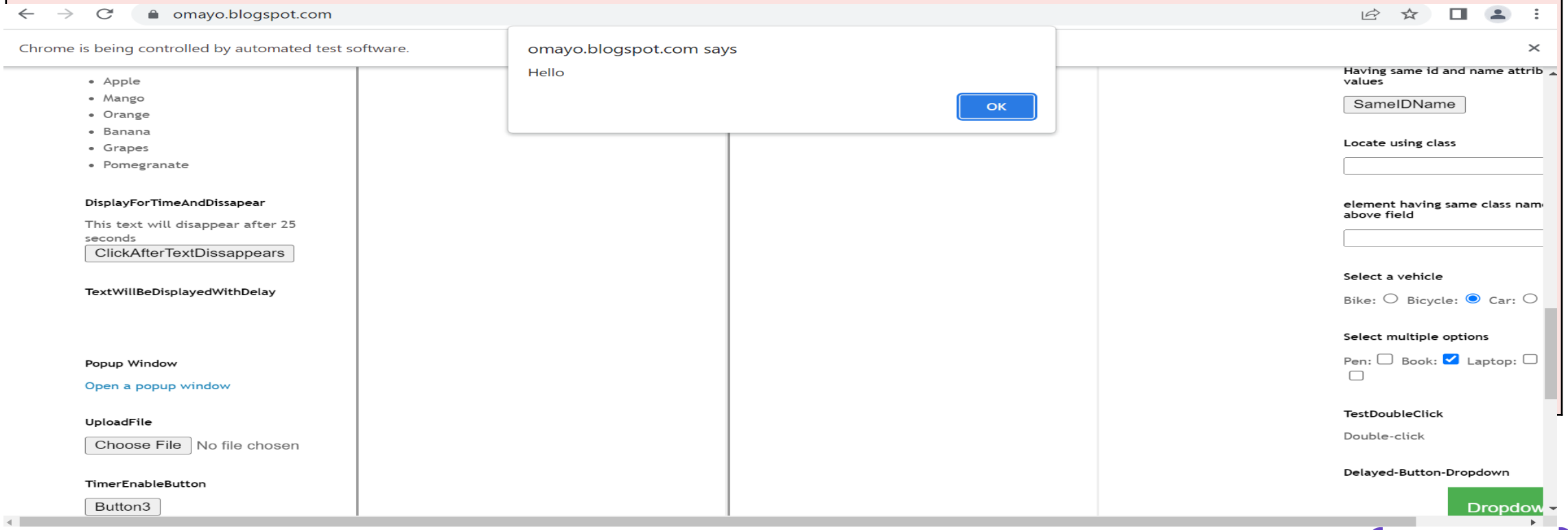
  wait.**until**(ExpectedConditions.visibilityOfElementLocated(By.cssSelector(".classlocator")));

An Example Scenario

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
public class Explicit {
 public static void main(String[] args) throws InterruptedException {
 System.setProperty("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");
 WebDriver driver=new ChromeDriver();
 WebDriverWait wait= new WebDriverWait(driver,Duration.ofSeconds(5));
driver.manage().window().maximize();
driver.get("https://omayo.blogspot.com/");
```

An Example Scenario

```
WebElement timerbutton =
wait.until(ExpectedConditions.elementToBeClickable(By.id("timerButton")));
timerbutton.click();
}}
```



Comparison between Implicit and Explicit Wait

Implicit Wait versus Explicit Wait

IMPLICIT WAIT	EXPLICIT WAIT
It is by default applied to all the elements in the script.	It is applicable to only a certain element which is specific to a certain condition.
We cannot wait based on a specified condition like element selectable/clickable unlike explicit.	In explicit, we can specify the wait based on a specific condition.
It is usually used when you are sure the element may be visible in a certain time	It is usually used, when you are not aware of the time of the element visibility. It is subjected to dynamic nature.

Fluent Wait

Fluent Wait

- The Fluent wait is similar to explicit wait in terms of its functioning.
- In Fluent wait, you perform a Selenium wait for an element when you are not aware of the time it may take to be visible or clickable.
- The few differential factors that Fluent wait offers are:
- **The polling frequency:**
  - In case of Explicit wait, this polling frequency is by default 500 milliseconds.
  - Using Fluent wait, you can change this polling frequency based on your needs,

    i.e you can tell your script to keep checking on an element after every ‘x’ seconds.

Fluent Wait

- **Syntax:**

```
Wait<WebDriver> fluentWait = new FluentWait<WebDriver>(driver)
.withTimeout(60, SECONDS) // this defines the total amount of time to wait for
.pollingEvery(2, SECONDS) // this defines the polling frequency
.ignoring(NoSuchElementException.class); // this defines the exception to ignore

 WebElement foo = fluentWait.until(new Function<WebDriver, WebElement>() {
 public WebElement apply(WebDriver driver)  //in this method defined your own subjected conditions for which we need to wait for
{
return driver.findElement(By.id("foo"));
}});
```

Fluent Wait

- **Ignore Exception :**

  - During polling, in case you do not find an element, you can ignore any exception like 'NoSuchElement' exception etc.

- Fluent Wait commands are most useful when interacting with web elements that can take longer durations to load. This is something that often occurs in Ajax applications.
- Fluent waits are also sometimes called smart waits because they don't wait out the entire duration defined in the code. Instead, the test continues to execute as soon as the element is detected – as soon as the condition specified in .until(YourCondition) method becomes true.

An Example Scenario

```
import java.util.concurrent.TimeUnit;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
public class Fluent {
 public static void main(String[] args) throws InterruptedException {
 System.setProperty("webdriver.chrome.driver", ".\\Driver\\chromedriver.exe");
 WebDriver driver=new ChromeDriver();
 driver.manage().window().maximize();
 driver.get("https://omayo.blogspot.com/");
 driver.findElement(By.className("dropbtn")).click();
```

An Example Scenario

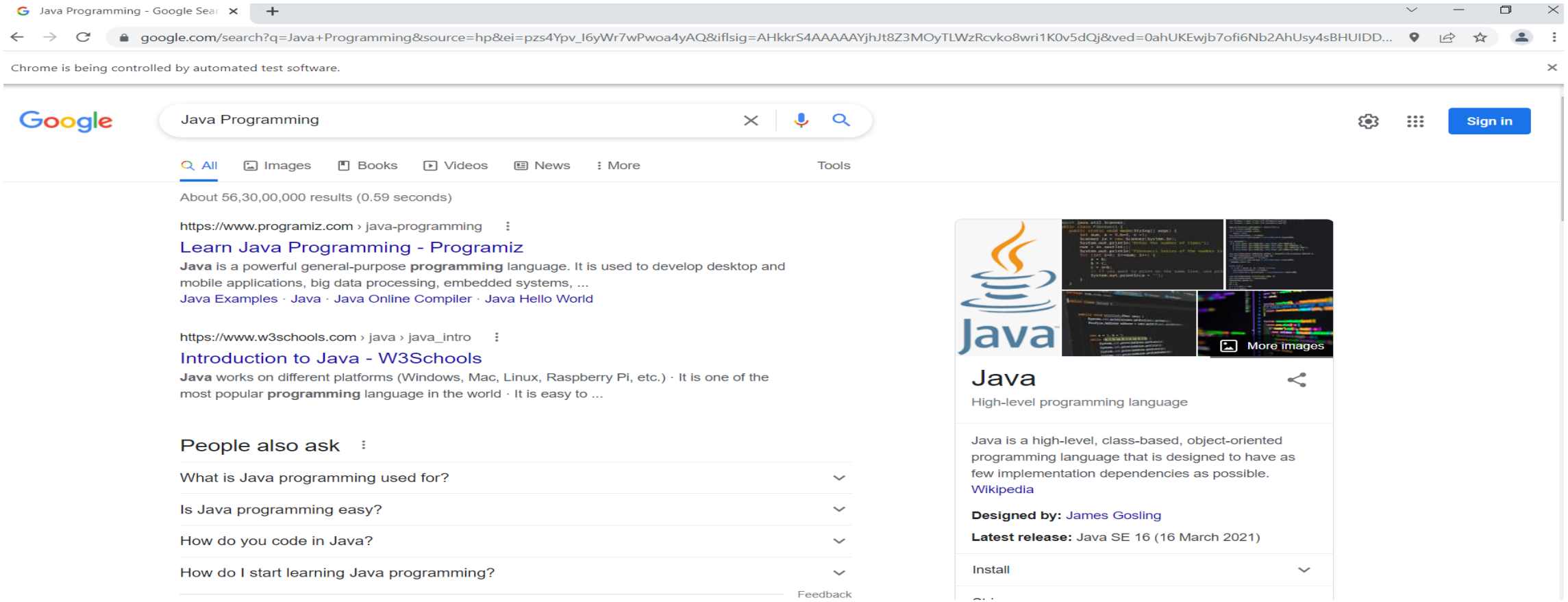
```
// Waiting 30 seconds for an element to be present on the page, checking
// for its presence once every 1 seconds.
Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
.withTimeout(Duration.ofSeconds(30))
.pollingEvery(Duration.ofSeconds(1))
.ignoring(NoSuchElementException.class);
WebElement facebookoption = wait.until(new Function<WebDriver, WebElement>() {
public WebElement apply(WebDriver driver) {
return driver.findElement(By.linkText("Facebook"));
}
});
facebookoption.click();
}
}
```

Program 1

```
public class Opengoole {

public static void main(String[] args) {
// TODO Auto-generated method stub
        System.setProperty("webdriver.chrome.driver","E:\\Testing\\chromedriver.exe");
        WebDriver driver=new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);
        driver.get("http://www.google.com");
        WebElement searchBox=driver.findElement(By.name("q"));
        searchBox.sendKeys("Java Programming"+Keys.ENTER);
}

}
```

Types of waits in Selenium

Output



expleo

Program 2

```
package com.coh.Seleniumtut;
import org.openqa.selenium.Alert;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
public class Alertdemo1 {
public static void main(String[] args) throws InterruptedException {
// TODO Auto-generated method stub
System.setProperty("webdriver.chrome.driver","c:\\Drivers\\chromedriver.exe");
WebDriver driver=new ChromeDriver();
@SuppressWarnings("deprecation")
```

Program 2 (Contd.)

```
WebDriverWait wait1=new WebDriverWait(driver,20);
driver.get("http://www.leafground.com/pages/Alert.html");
WebElement Alertbox1=
driver.findElement(By.xpath("//*[@id='contentblock']/section/div[1]/div/div/button"));
Alertbox1.click();
Alert alert1=wait1.until(ExpectedConditions.alertIsPresent());
driver.switchTo().alert();
Thread.sleep(3000);
alert1.accept();
}
}
```

Output