Types of Selenium Wait Commands

**Implicit wait:** This is used to tell the WebDriver to wait for a certain amount of time before it throws a “**NoSuchElementException**”. This wait is applied globally, which means that the same wait mechanism will work for all the elements in the given code.

driver.manage().timeouts().implicitlyWait(20, TimeUnit.SECONDS);

In the above example, if the element is not located on the webpage within 20 secs, it will throw an exception automatically.

* **Explicit wait**: This is used to tell the WebDriver to wait for certain expected conditions or maximum time exceeded before throwing an “**ElementNotVisibleException**” exception. It is an intelligent kind of **wait** but can be applied only for specified elements.

WebDriverWait wait=new WebDriverWait(driver, 20);

wait.until(ExpectedConditions.visibilityOf(element));

In this case, Webdriver will wait explicitly for 20 secs for the element to be visible, and if it is not visible, it will throw an exception.

### Why and when to use Thread.sleep in Selenium?

Now, the question arises, if Selenium provides these two types of wait commands, why would an automation tester opt for using **Thread.sleep()**. It would be interesting to know the role of **Thread.sleep()** command.

Below are some of the scenarios where **Thread.sleep()** comes to the rescue when none of the wait mechanisms works.

#### **1.  Testing dynamic Web Elements**

Many web applications have a carousel implementation to display dynamic content. If you need to verify the visibility of the next item in the carousel, it is always better to use **Thread.sleep()** instead of Selenium waits as the carousel items are very dynamically displayed.

**Also Read:** [Selenium WebElement Commands](https://www.browserstack.com/guide/selenium-webelement-commands)

#### **2. Third-party testing**

Nowadays, almost every web application has integration with third-party websites. Along with testing AUT (Application Under Test), you also need to validate that after clicking on any third-party link, it navigates to the valid webpage, and the page loads properly.

To assert this behaviour it is wise to use **Thread.sleep()** as it is very unpredictable to verify the load time of any third-party website.

#### **3. Switching Windows or Tabs**

Clicking on some of the links on the webpage opens them in a new browser window tab or a new browser instance. In such cases, to perform browser automation tests, you need to switch the driver to the new browser window tab/ browser instance and perform actions on it.

When you switch and directly perform any automation event, it will throw **NoSuchElementException** as the WebDriver has just switched and it is unable to locate the element. Applying **Thread.sleep()** after the switch command helps to solve this.

Thread.Sleep in Selenium Java

While the above scenarios justify why **Thread.sleep()** command in Selenium Java is important. Let’s understand **Thread.sleep()** function in detail using some points below:

* As per the official definition from the [Oracle Java Documentation](https://docs.oracle.com/javase/tutorial/essential/concurrency/sleep.html), **Thread.sleep()** causes the current thread to suspend execution for a specified period.
* **Thread.sleep()** is not a Selenium wait, it is provided by Java.
* It suspends the code for the specified time and can be useful in debugging the script under test.

Thread class is a class present in **Java.lang** package that is basically a thread of execution of the programs.

Thread class has two overloaded sleep methods, **Thread.sleep(long millis)** and **Thread.sleep(long millis, int nanos).**

* **Thread.sleep(long millis)**

**Thread.sleep(2000):** Duration is in milliseconds for which thread will sleep.

* **Thread.sleep(long millis, int nanos)**

**Thread.sleep(2000, 2000):** The second parameter is an additional time in nanoseconds for which thread will sleep. It ranges from 0 to 999999.

* **Thread.sleep()** does not have a return type, and it returns void.

**Thread.sleep** methods throws **InterruptedException** when any other thread interrupts the current thread and should be handled by the throws method or try catch block.

try {

Thread.sleep(2000);

} catch (InterruptedException e) {

// TODO Auto-generated catch block

e.printStackTrace();

}

OR

public void threadTest() throws InterruptedException {

Thread.sleep(2000);

}

* It throws an **IllegalArgumentException** when sleep time is negative.

Thread.sleep(-2000;)

* Both the **sleep()**methods of Thread class are static methods. Hence, they should be accessed in a static way by using class name.

Thread.sleep(2000);

### Limitations of Thread.sleep

Using **Thread.sleep()** frequently in an automation framework is not a good practice. If the applied sleep is of 5 secs and the web element is displayed in 2 secs only, the extra 3 secs will increase the execution time. And if you use it more often in the framework, the execution time would increase drastically.

You always have to guess and apply **Thread.sleep()** seconds in advance, as there is no guarantee that the web element would be discoverable under that specified time.

**Note:**

Implicit and thread sleep wait increases test case run time. Suppose time provided is 10 sec and if ele is found in 2 secs only, still driver will wait for leftout 10 seconds.

Implicit wait is applicablw to all webelements if applied in program

Explicit wait is for specific elements. And if ele is found in 2 sec it will execute next lines without waiting.