🌟 Synchronization In Selenium

In **Selenium**, **Synchronization** means **making your test script wait for the web elements to be ready before performing actions** (like click, sendKeys, etc.).

It ensures your test runs **in sync with the speed of the application** — not too fast, not too slow.

🧭 Why Synchronization Is Needed

Web applications load elements dynamically — sometimes due to:

- **AJAX calls** - **Asynchronous JavaScript and XML** It's a technique that lets a webpage **send or receive data from the server in the background without reloading the whole page, just the required part gets updated without a full page reload.**
- **JavaScript execution**
- **Network latency**

If Selenium tries to interact before an element is ready → it throws exceptions like:

- NoSuchElementException
- ElementNotVisibleException
- ElementNotInteractableException

🕒 1. Thread.sleep()

🔹 **Description:**

A Java method that **pauses execution for a fixed time**, regardless of whether the element is ready or not.

Thread.sleep(3000); // Waits for 3 seconds

⚠️ **Drawbacks:**

- Unconditional delay — slows down the test even if elements are ready early.
- Makes test execution inefficient.

🧠 **Not Recommended** — Use only for debugging or temporary testing.

⏱️ 2. Implicit Wait

🔹 **Description:**

Tells Selenium WebDriver to **wait for a certain maximum time** while searching for an element before throwing a NoSuchElementException.

Once declared, it applies **globally** to all element searches.

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));

🔹 **How It Works:**

1. Selenium starts searching for the element in the DOM.
2. If found immediately → proceeds without waiting.
3. If not found → keeps checking until:
    - Element appears within the timeout period ✅
    - Timeout expires ❌ → throws NoSuchElementException

🔹 **Example:**

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5));

driver.findElement(By.id("loginBtn")).click();

If the button appears in 3 seconds, Selenium proceeds immediately — it won't wait the full 5 seconds.

🔹 **Key Points:**

Concept	**Explanation**
Scope	Applies to all findElement calls
Polling	Checks DOM periodically (milliseconds interval)
Wait Time	Maximum waiting time (not fixed delay)
Reusability	Remains active until changed or driver quits
Flexibility	Low — same time for all elements

⚠️ **Drawbacks:**

- Applies to **all elements** equally.
- Can **hide performance issues** — slow pages may pass silently.
- Cannot wait for specific conditions like *visibility* or *clickability*.

🎯 **Example Scenario: Flight Search**

Small City (Loads Fast)

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(5))

✅ Works fine for small routes loading within 5 seconds.

Large Route (Loads Slow)

You increase it to 15 seconds:

driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(15));

✅ Works for large routes.

⚠️ But now every element will wait up to 15 seconds, if a small city that is assumed to load in 5 seconds takes 10 seconds still the test will not fail — **slowing tests** and **hiding slow performance issues**.

🧩 3. Explicit Wait

🔹 **Description:**

Explicit Wait pauses the execution **until a specific condition** is met for a **particular element**.

It gives precise control — you can wait for:

- element visibility
- element to be clickable
- presence of text
- alerts, frames, etc.

Two Types:

1. **WebDriverWait**
2. **FluentWait**

🧠 1. WebDriverWait

A specialization of FluentWait that waits until a condition is met or timeout occurs.

🔹 **Syntax:**

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));

🔹 **Working Mechanism:**

1. Waits up to 10 seconds for the element.
2. Checks DOM every **500 milliseconds (default polling)**.
3. Proceeds immediately if condition met.
4. Throws TimeoutException if not met in time.

🔹 **Common Expected Conditions:**

Condition	**Description**
visibilityOfElementLocated()	Wait until element is visible
elementToBeClickable()	Wait until element is clickable
presenceOfElementLocated()	Element present in DOM (not necessarily visible)
textToBePresentInElement()	Wait until element contains specific text
alertIsPresent()	Wait until an alert appears
invisibilityOfElementLocated()	Wait until element disappears
frameToBeAvailableAndSwitchToIt()	Wait for a frame and switch to it

---

🔹 **Example:**

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

WebElement loginBtn =
wait.until(ExpectedConditions.elementToBeClickable(By.id("loginButton")));

loginBtn.click();
```

✅ Stops waiting as soon as element is clickable — no unnecessary delay.

🔹 **Key Points**

Concept	**Description**
Scope	Specific element(s)
Condition-based	Waits for visibility/clickability/text etc.
Polling frequency	500ms (default)
Stops early	Exits immediately if condition met
Exception	Throws TimeoutException if not met
Recommended	For dynamic, AJAX-based elements

🔹 **WebDriverWait vs Implicit Wait**

Feature	**Implicit Wait**	**Explicit Wait**
Applies To	All elements	Specific element(s)
Type	Fixed global time	Condition-based
Flexibility	Low	High
Error on Failure	NoSuchElementException	TimeoutException
Use Case	Stable pages	Dynamic pages

⚙️ 2. Fluent Wait

🔹 **Description:**

Fluent Wait is an advanced form of **Explicit Wait** that provides **fine-grained control** over:

- ⏱️ **Timeout duration** (maximum waiting time)
- 🔁 **Polling frequency** (how often to check for the condition)
- 🚫 **Ignored exceptions** (like NoSuchElementException)

It continuously polls the DOM for the desired condition **until either**:

- The element becomes available and visible ✅
- The timeout expires ❌

🔹 **Syntax:**

```
Wait<WebDriver> wait = new FluentWait<>(driver)
  .withTimeout(Duration.ofSeconds(10))
  .pollingEvery(Duration.ofSeconds(2))
  .ignoring(NoSuchElementException.class);
```

Implementation to put on a webelement -

```
WebElement element = wait.until(new Function<WebDriver, WebElement>()

{

public WebElement apply(WebDriver driver)

        {

                return driver.findElement(By.id("username"));

        }

});
```

🔹 **Fluent Wait vs WebDriverWait**

Feature	**WebDriverWait**	**FluentWait**
Polling	Every 500 ms (default)	Customizable interval (e.g., 2s, 5s, etc.)
Control	Limited configuration	Full control over timeout, polling, and exceptions
Implementation	Extends FluentWait	Base class implementing Wait Interface

---

🔹 **Practical Example — Dynamic Element Handling**

```
// Click the Start button

driver.findElement(By.cssSelector("div#start button")).click();


// Create FluentWait instance

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

            .withTimeout(Duration.ofSeconds(10))   // Total wait time

           .pollingEvery(Duration.ofSeconds(6))   // Check every 6 seconds

        .ignoring(NoSuchElementException.class); // Ignore missing elements
```

```
// Define custom wait condition
WebElement f = wait.until(new Function<WebDriver, WebElement>() {
  public WebElement apply(WebDriver driver) {
    // Check if element is visible
    WebElement element = driver.findElement(By.cssSelector("div#finish h4"));
    if (element.isDisplayed()) {
      return element; // Return once visible
    } else {
      return null; // Keep waiting
    }
  }
});
// Print the text once element is visible
System.out.println(f.getText());
```

🧠 **Explanation:**

1. **withTimeout()** → Defines the maximum time to wait.
2. **pollingEvery()** → Frequency at which Selenium checks for the element.
3. **ignoring()** → Ignores specified exceptions during polling.
4. **until()** → Accepts a custom function that returns a WebElement once the condition is met.
5. If the element is not visible even after the timeout — TimeoutException is thrown.

💡 **Key Points:**

- FluentWait is most useful for **highly dynamic or AJAX-based applications**.
- It provides **maximum flexibility** for custom waiting logic.
- Prefer FluentWait when you need **conditional visibility** or **custom retry logic**.

🔹 **Example Behavior**

If timeout = **10 seconds** and polling = **2 seconds**:

Time (sec)	WebDriverWait	FluentWait
1	Checks DOM	Waits
2	Checks DOM	Checks
3	Checks DOM	Waits
4	Checks DOM	Checks
...	Every ms	Every 2 sec

🧠 If an element appears at **3rd second**:

- **WebDriverWait** detects immediately (active monitoring).
- **FluentWait** detects only at **4th second** (next polling interval).

🧩 **Real-Life Use Case:**

Scenario:

An e-commerce site after payment shows:

1. “Your Card Is Accepted” (3rd second)
2. “Your Order is Being Processed” (7th second)

Both messages share **same HTML properties**, so the second message cannot be differentiated easily using WebDriverWait — it may detect the first one.

✅ **Solution:**

Use **Fluent Wait** with a polling interval of **4 seconds**:

- At 3s: “Card Accepted” appears.
- FluentWait checks at 4s → skips it.
- Next check at 8s → detects “Order Processed” message successfully.

🔹 **Advantages:**

- Gives **granular control** over polling time and exception handling.
- Useful for **unpredictable** or **slow-changing** elements.

⚠️ **Disadvantages:**

- Slightly **complex** and **verbose** (more lines of code).
- Slower if polling interval too large.

🧠 **Summary**

Wait Type	**Applies To**	**Control**	**Use Case**	**Recommended**
Thread.sleep()	Entire script	❌ None	Debug/temporary wait	❌ No
Implicit Wait	All elements	⚠️ Limited	Stable elements	⚠️ Sometimes
Explicit Wait (WebDriverWait)	Specific elements	✅ Good	Dynamic elements	✅ Yes
Fluent Wait	Specific elements	✅ Highest	Complex dynamic flows	✅ Advanced

🔹 **Note:**

Both WebDriverWait and FluentWait implement the Wait interface in Selenium.