# **Browser Waits &**

# **Synchronization in Selenium**

## **⏳ Types of Waits in Selenium**

| **Wait Type** | **Description** | **When to Use** | **Pros** | **Cons** |
| --- | --- | --- | --- | --- |
| Implicit Wait | Global wait applied to all elements | Simple scripts with consistent load times | Easy to implement | Slows down entire test suite |
| Explicit Wait | Condition-based wait for specific elements | Dynamic content, AJAX-heavy pages | Precise control | More code required |
| Fluent Wait | Configurable wait with polling frequency | Complex scenarios needing custom polling | Highly flexible | Complex implementation |

## **1. Implicit Wait**

java

// Sets a global timeout for element location

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

* Applies to: All findElement() calls
* Behavior: Polls DOM until element is found or timeout expires
* Best Practice: Avoid mixing with explicit waits

## **2. Explicit Wait (Most Recommended)**

java

WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(20));

// Common Expected Conditions

wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("login")));

wait.until(ExpectedConditions.elementToBeClickable(By.cssSelector(".btn")));

wait.until(ExpectedConditions.presenceOfElementLocated(By.name("q")));

### **Common ExpectedConditions**

| **Condition** | **Description** | **Use Case** |
| --- | --- | --- |
| visibilityOfElementLocated() | Element is visible AND present in DOM | Checking UI elements |
| elementToBeClickable() | Element is enabled and clickable | Button interactions |
| presenceOfElementLocated() | Element exists in DOM (may be hidden) | Form field validation |
| textToBePresentInElement() | Element contains specific text | Dynamic content loading |
| alertIsPresent() | Alert dialog appears | Popup handling |

## **3. Fluent Wait (Advanced)**

java

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

.withTimeout(Duration.ofSeconds(30))

.pollingEvery(Duration.ofSeconds(2))

.ignoring(NoSuchElementException.class));

WebElement element = wait.until(d -> {

WebElement e = d.findElement(By.id("dynamic"));

return e.isDisplayed() ? e : null;

});

### **FluentWait Configuration**

| **Method** | **Purpose** | **Default** |
| --- | --- | --- |
| withTimeout() | Maximum wait duration | Required |
| pollingEvery() | Frequency of checks | 500ms |
| ignoring() | Exceptions to ignore | None |

## **🚀 Best Practices**

1. Prefer explicit waits over implicit waits (90% of cases)
2. Never mix implicit and explicit waits (causes unpredictable delays)
3. Use reasonable timeouts (10-30s for most web apps)
4. Combine with Page Objects for maintainable code
5. Custom expected conditions for complex scenarios:

java

wait.until(driver -> {

return ((JavascriptExecutor)driver)

.executeScript("return jQuery.active == 0"));

});

## **💡 Pro Tips**

* For Angular/React apps, use specialized waits:

java

// Angular

wait.until(ExpectedConditions.jsReturnsValue("return angular.element(document).injector().get('$http').pendingRequests.length === 0"));

* For file downloads, combine with file system checks:

java

wait.until(d -> new File("/downloads/file.zip").exists());