Material descargable

egg

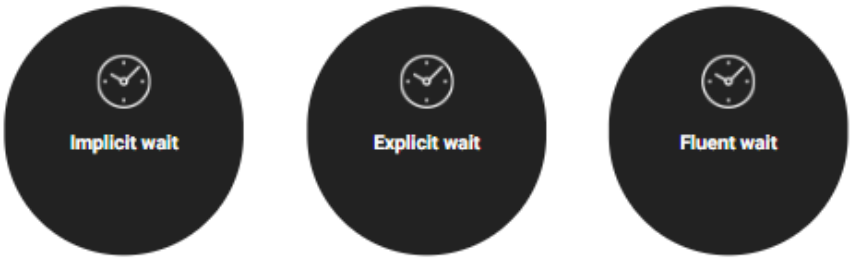
WEB AUTOMATION

¿Qué son los waits en Selenium?

Los **waits** (esperas) en Selenium son una técnica utilizada para gestionar el tiempo que debe esperar un script antes de interactuar con un elemento en la página. Esto es crucial en pruebas automatizadas, ya que las aplicaciones web modernas pueden tener tiempos de carga variables, y si un script intenta interactuar con un elemento antes de que esté disponible, se pueden producir errores como **NoSuchElementException**.

En Selenium, los **waits** ayudan a que los scripts sean más estables al sincronizarse con los tiempos de carga y garantizar que los elementos estén listos para ser manipulados antes de intentar interactuar con ellos.

¿Cómo se clasifican los waits?



Existen tres tipos principales de **waits** (esperas) en Selenium:

- **Implicit Wait (Espera implícita)**: La espera implícita es configurada una sola vez en el WebDriver y se aplica a todos los elementos. Son ajustes globales que se aplican a la instancia de tu WebDriver. Define un tiempo máximo durante el cual Selenium esperará para encontrar un elemento antes de lanzar una excepción. Si el elemento no se encuentra dentro de ese tiempo, se lanzará una excepción **NoSuchElementException**. Esta espera es útil para elementos que tardan en cargarse de forma aleatoria en la página.

```
// Para Selenium 3.x o versiones anteriores
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS); //
Espera implícita de 10 segundos

// Para Selenium 4 y versiones posteriores
driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10)); //
Espera implícita de 10 segundos usando Duration
```

- **Explicit Wait (Espera explícita)**: La espera explícita es utilizada para esperar a que se cumpla una condición específica antes de continuar con la ejecución. A diferencia de la espera implícita, la espera explícita se aplica solo a un elemento o condición particular, lo que permite un control más preciso. Es especialmente útil para situaciones en las que un elemento puede tardar en ser interactuable, como en el caso de animaciones o la aparición de un pop-up.

```
// Selenium ofrece la clase WebDriverWait, junto con una variedad de
condiciones esperadas (ExpectedConditions), // para implementar
esperas explícitas de manera efectiva y precisa.
WebDriverWait wait = new WebDriverWait(driver, 10);
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("button")));
```

- **Fluent Wait (Espera fluida)**: La espera fluida es una forma avanzada de espera explícita. Permite configurar el intervalo entre las verificaciones repetidas y el tiempo máximo de espera. Es útil cuando necesitas realizar verificaciones continuas y personalizar la espera según las necesidades del contexto, como en el caso de elementos que cambian de estado constantemente.

```
// Este código utiliza FluentWait en Selenium 4 o posterior
Wait<WebDriver> wait = new FluentWait<>(driver)
    .withTimeout(30, TimeUnit.SECONDS)
    .pollingEvery(5, TimeUnit.SECONDS)
    .ignoring(NoSuchElementException.class);
```

Implicit Wait

Las **esperas implícitas** permiten establecer un tiempo de espera global para todos los elementos de una página. Una vez configurada, Selenium esperará el tiempo establecido antes de lanzar una excepción si no encuentra un elemento.

Características:

- **Globalidad**: Una vez que estableces una espera implícita, esta se aplica durante toda la vida del objeto WebDriver. Esto significa que afecta a todos los elementos de la página web.
- **Sencillez**: Se configura fácilmente con un solo método.
- **Flexibilidad**: Se puede modificar en cualquier parte del script.
- **Limitaciones**: No es adecuada para condiciones específicas, como esperar que un elemento sea visible o clickeable.
- **Uso Recomendado**: Usar con precaución, ya que puede hacer que las pruebas sean más lentas si se establece un tiempo largo, especialmente para múltiples elementos.

Ejemplo de implementación:

```
public class Implicitas {
    public static void main(String[] args) {
       // Establece la propiedad del controlador de Chrome para usar el navegador
adecuado
       System.setProperty("webdriver.chrome.driver",
"/ruta/del/controlador/chromedriver");

       // Crea una instancia del navegador Chrome
       WebDriver driver = new ChromeDriver();

       // Configura una espera implícita de 2 segundos
       // Esto le dice a Selenium que espere hasta 2 segundos antes de lanzar una
excepción
       // si no encuentra el elemento especificado
       driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(2));

       // Navega a la página de Google
       driver.get("https://www.google.com");

       // Intenta encontrar un elemento en la página con el id "unElemento"
       // Si el elemento no se encuentra en 2 segundos, Selenium lanzará una
excepción
       WebElement element = driver.findElement(By.id("unElemento"));

       // Continuar con las interacciones o validaciones necesarias después de
encontrar el elemento
       // Por ejemplo: element.click() o validar contenido del elemento
```

```
        // Cierra el navegador
        driver.quit();
    }
}
```

⚠️**Importante: Actualización en la Configuración de Tiempo de Espera Implícito en Selenium**

La configuración del tiempo de espera en Selenium ha sido actualizada en versiones recientes. Es importante que estén al tanto de estos cambios para mantener su código actualizado y conforme a las mejores prácticas.

Cambios en la Configuración de Tiempo de Espera Implícito:

Versión Anterior (Selenium 3):

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

- **En Selenium 3, se utilizaba el método `implicitlyWait` con `TimeUnit` para definir el tiempo de espera implícito.**

Versión Actual (Selenium 4):

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

- **En Selenium 4, el método `implicitlyWait` se actualizó para utilizar la clase `Duration` de Java 8 y posteriores.**
- **Este cambio hace que el código sea más legible y consistente con el manejo de tiempos en Java.**

Importancia del Cambio:

- **No Deprecado: El método `implicitlyWait` no ha sido deprecado, simplemente se ha actualizado la forma de definir la duración del tiempo de espera.**
- **Mejoras en la Legibilidad: El uso de `Duration` mejora la legibilidad del código y la consistencia con otras partes del lenguaje Java.**

Mantenerse al día con estos cambios es crucial para asegurar la funcionalidad y eficiencia de sus pruebas automatizadas. ¡Asegúrense de actualizar su código según estas nuevas prácticas!

Explicit Wait

Las **esperas específicas** permiten esperar condiciones específicas antes de continuar con la ejecución del script. A diferencia de las esperas implícitas, se aplica a situaciones concretas, como cuando un elemento debe ser visible o en el que se puede hacer clic.

- **Precisión**: Permiten especificar condiciones exactas bajo las cuales quieres que continúe la ejecución del script. Esto significa que puedes esperar a que cierto elemento sea visible, esté presente en el DOM o hasta que cumpla alguna otra condición específica.
- **Flexibilidad**: Puedes aplicar diferentes esperas para diferentes elementos o condiciones en el mismo script, lo que te da una gran flexibilidad para manejar variaciones en los tiempos de carga o respuesta de la página.
- **Eficiencia**: Al esperar solo las condiciones necesarias antes de proceder, las esperas explícitas pueden reducir los tiempos de espera innecesarios y, por lo tanto, acelerar la ejecución de tus pruebas.
- **Personalización**: Ofrecen la posibilidad de definir condiciones personalizadas, lo que te permite esperar por escenarios muy específicos que no están cubiertos por las condiciones predefinidas de Selenium.

Algunos ejemplos de condiciones esperadas:

alertIsPresent()	Comprueba si hay una alerta presente.
elementToBeClickable()	Espera a que un elemento sea clicable.
elementToBeSelected()	Espera a que un elemento esté seleccionable.
frameToBeAvaliableAndSwitchToIt()	Espera a que un marco esté disponible y cambia al mismo.
invisibilityOfTheElement()	Espera a que un elemento desaparezca.
invisibilityOfElementWithText()	Espera a que un elemento con un texto específico desaparezca.
presenceOfAllElements()	Espera a que todos los elementos estén presentes.

presenceOfElement()	Espera a que un elemento esté presente.
textToBePresentInElement()	Espera a que un texto esté presente en un elemento.
titleContains()	Espera a que el título de la página contenga cierto texto.
visibilityOf()	Espera a que un elemento sea visible.
visibilityOfAllElements()	Espera a que todos los elementos sean visibles.

¿Cómo se programan las esperas explícitas?

Para utilizar las esperas explícitas en Selenium, necesitas familiarizarte con dos componentes clave: WebDriverWait y ExpectedConditions:

- **Instanciar WebDriverWait:** crea una instancia del "esperador", especificando el WebDriver y el tiempo máximo de espera:

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

- **Especificar la Condición de Espera:** Después, utilizas el objeto wait para esperar una condición específica. Selenium proporciona una clase ExpectedConditions con métodos estáticos para las condiciones de espera más comunes:
    - Esperar a que un elemento sea clickeable:

```
wait.until(ExpectedConditions.elementToBeClickable(By.id("someId")));
```

    - Esperar a que un elemento sea visible:

```
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("some")));
```

    - Esperar a que un elemento desaparezca:

```
wait.until(ExpectedConditions.invisibilityOfElementLocated(By.id("some")));
```

- **Definir Condiciones Personalizadas**: Además de las condiciones predefinidas, puedes crear tus propias condiciones personalizadas utilizando ExpectedCondition y expresiones lambda en Java. Esto te permite esperar por condiciones muy específicas que no están cubiertas por los métodos de ExpectedConditions.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
wait.until(new ExpectedCondition<Boolean>() {
    public Boolean apply(WebDriver driver) {
        return driver.getTitle().equals("Expected Title");
    }
});
```

Al combinar WebDriverWait con ExpectedConditions, obtienes un control granular y eficiente sobre las esperas en tus pruebas, permitiéndote esperar exactamente por lo que necesitas antes de proceder. Esto no solo hace que tus pruebas sean más robustas y confiables, sino que también optimiza los tiempos de ejecución al evitar esperas innecesarias.

Ejemplo de implementación::

```
public class Explicitas {
    public static void main(String[] args) {
       System.setProperty("webdriver.chrome.driver", "/ruta");
       WebDriver driver = new ChromeDriver();
       driver.get("https://www.google.com");

       // Actualización para Selenium 4: Usar Duration directamente
       WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

       // Esperar hasta que el elemento sea visible
       WebElement element = wait.until(
          ExpectedConditions.visibilityOfElementLocated(By.id("unElemento"))
       );

       // Ejemplo de una condición personalizada
       wait.until(new ExpectedCondition<Boolean>() {
          public Boolean apply(WebDriver driver) {
             return driver.getTitle().equals("Expected Title");
          }
       });

       // Continuar con las interacciones o validaciones necesarias después de encontrar
el elemento
       // Por ejemplo: element.click() o validar contenido del elemento

       // Cierra el navegador
       driver.quit();
    } }
```

En este ejemplo, el WebDriver esperará hasta 10 segundos para que el elemento con el ID "unElemento" sea visible. Si el elemento se vuelve visible antes de que transcurran los 10 segundos, el código continuará ejecutándose sin esperar todo el tiempo establecido. Si el elemento no se vuelve visible en 10 segundos, se lanzará una excepción de tiempo de espera (TimeoutException).

Fluent Wait

Las **esperas fluidas** permiten gestionar las esperas de manera más flexible y granular, como especificar el tiempo de espera máximo, la frecuencia con la que verificar la condición y qué excepciones ignorar.

- **Personalización del Tiempo de Espera**: Con las esperas fluidas, tú decides cuánto tiempo máximo esperar por una condición antes de que se lance una excepción. Esto te da control sobre el rendimiento de tus pruebas.
- **Frecuencia de Verificación**: Puedes establecer la frecuencia con la que FluentWait debe verificar la condición que estás esperando. Esto es útil para no sobrecargar el DOM o la aplicación con verificaciones constantes.
- **Ignorar Excepciones Específicas**: FluentWait te permite especificar una o varias excepciones para ignorar mientras espera a que se cumpla una condición. Esto es especialmente útil cuando esperas que se produzcan y manejen ciertas excepciones, como NoSuchElementException durante la espera.
- **Uso de Condiciones Personalizadas**: A diferencia de las esperas implícitas y explícitas, las esperas fluidas te permiten utilizar tus propias condiciones personalizadas o las proporcionadas por el ExpectedConditions para esperar elementos o condiciones específicas.

¿Cómo se programan las esperas fluidas?

Programar esperas fluidas en Selenium te permite tener un control granular sobre cómo y cuándo se espera por elementos o condiciones específicas. Los métodos más comunes son:

withTimeout(Duration duration)	Establece la cantidad máxima de tiempo para esperar una condición. La espera

	terminará y lanzará una excepción si la condición no se cumple dentro del período de tiempo especificado.
pollingEvery(Duration duration)	Establece la frecuencia con la que se debe verificar la condición.
ignoring(Class<? extends Throwable> exceptionType)	Le dice a Fluent Wait que ignore ciertas excepciones mientras espera que se cumpla una condición.
until(Function<? super T, V> isTrue)	Este método acepta una instancia de Function que debe devolver un valor diferente de null o false si la condición se cumple.
withMessage(String message)	Este método permite establecer un mensaje personalizado que se incluirá en la excepción si se alcanza el tiempo de espera sin que se cumpla la condición.
until(Predicate<T> isTrue)	Similar a until(Function<? super T, V> isTrue), pero acepta un Predicate y continúa la espera hasta que el predicado devuelva true.

Ejemplo de implementación:

```
public class Fluidas {
    public static void main(String[] args) {
       System.setProperty("webdriver.chrome.driver", "/ruta");
       WebDriver driver = new ChromeDriver();
       driver.get("https://www.google.com");

       // Espera fluida
       Wait<WebDriver> wait = new FluentWait<WebDriver>(driver)
          .withTimeout(Duration.ofSeconds(10)) // Tiempo máximo
de espera
          .pollingEvery(Duration.ofMillis(500)) // Frecuencia de
verificación
          .ignoring(NoSuchElementException.class);

       // Esperar hasta que el elemento "search" esté presente
       WebElement search = wait.until(new Function<WebDriver,
WebElement>() {
          public WebElement apply(WebDriver driver) {
             return driver.findElement(By.id("search"));
```

```
            }
        });

        // Asegúrate de que loginButton sea reemplazado con el
elemento correspondiente
        // Por ejemplo, si estás buscando un botón de búsqueda en
Google:
        WebElement searchButton =
driver.findElement(By.name("btnK"));
        searchButton.click(); // Aquí se realiza la acción en el
botón

        driver.quit();
    }
 }
```

En este ejemplo, si el elemento con el ID "search" no está disponible de inmediato, WebDriver esperará hasta 10 segundos, verificando su presencia cada 500 milisegundos. Si el elemento se encuentra disponible antes de que transcurran los 10 segundos, el código continuará ejecutándose sin esperar todo el tiempo establecido. Las esperas fluidas te dan la flexibilidad de adaptar las esperas a las necesidades específicas de tus pruebas, mejorando su eficiencia y efectividad. Al entender y aplicar correctamente estas técnicas, mejorarás significativamente la calidad de tus pruebas automatizadas con Selenium.