Selenium Day 2 - Classroom Session - Trainer's Handbook

Session Agenda

1. **Introduction to Synchronization in Selenium**
2. **Types of Waits in Selenium**
    - Implicit Wait
    - Explicit Wait
    - Fluent Wait
3. **Hands-on Implementation of Waits**
4. **Best Practices & Common Interview Questions**
5. **Post-Classroom Assignment (2 Hours)**
6. **Expected Outcomes by End of Day 2**

Introduction to Synchronization in Selenium

What is Synchronization?

Synchronization ensures that the Selenium script waits for the web elements to load before performing actions on them.

Why is Synchronization Needed?

- Web elements **may take time** to load due to network speed, browser rendering, or dynamic page updates.

- **Hardcoded waits (Thread.sleep)** lead to unnecessary delays and inefficient test execution.
- Proper synchronization **optimizes test execution time** and prevents flaky tests.

Types of Synchronization in Selenium:

1. **Unconditional Waits** – Thread.sleep(), which is not recommended.
2. **Conditional Waits** – Implicit, Explicit, and Fluent Waits (Preferred).

---

2. Types of Waits in Selenium

A. Implicit Wait

Definition: Implicit Wait tells WebDriver to wait for a specified amount of time before throwing a NoSuchElementException. It is applied globally to all web elements in a test.

Syntax:

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

Example:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;
import java.time.Duration;

public class ImplicitWaitExample {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10));

        driver.get("https://example.com");
        WebElement loginButton = driver.findElement(By.id("login"));
        loginButton.click();

        driver.quit();
    }
}
```

Key Points:

- ✔ Applied once and works for the entire session.
- ✔ Applies to all findElement() calls.
- ✔ Can lead to unnecessary delays if not used properly.

B. Explicit Wait

Definition: Explicit Wait waits for a specific condition to occur before proceeding further. It uses the WebDriverWait class along with ExpectedConditions.

Syntax:

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));
```

Example:

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.WebDriverWait;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.By;
import java.time.Duration;

public class ExplicitWaitExample {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));
        WebElement usernameField = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username")));

        usernameField.sendKeys("testUser");

        driver.quit();
    }
}
```

Key Points:

- ✔ **More efficient than Implicit Wait**, as it waits only for specific elements.
- ✔ Applied only to the required elements.
- ✔ Supports multiple ExpectedConditions:
  - visibilityOfElementLocated(By locator)
  - elementToBeClickable(By locator)
  - textToBePresentInElement(By locator, "expectedText")

C. Fluent Wait

Definition: Fluent Wait is an advanced version of Explicit Wait that allows polling at regular intervals before throwing an exception.

Syntax:

```
Wait<WebDriver> wait = new FluentWait<>(driver)
        .withTimeout(Duration.ofSeconds(30))
        .pollingEvery(Duration.ofSeconds(5))
        .ignoring(NoSuchElementException.class);
```

Example:

```java
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.FluentWait;
import org.openqa.selenium.By;
import java.time.Duration;
import java.util.function.Function;

public class FluentWaitExample {
    public static void main(String[] args) {
        WebDriver driver = new ChromeDriver();
        driver.get("https://example.com");

        FluentWait<WebDriver> wait = new FluentWait<>(driver)
                .withTimeout(Duration.ofSeconds(20))
                .pollingEvery(Duration.ofSeconds(2))
                .ignoring(NoSuchElementException.class);

        WebElement submitButton = wait.until(new Function<WebDriver, WebElement>() {
            public WebElement apply(WebDriver driver) {
                return driver.findElement(By.id("submit"));
            }
        });

        submitButton.click();
        driver.quit();
    }
}
```

Key Points:

- Polls the DOM at regular intervals.
- Useful for handling **dynamically loaded elements**.
- Can ignore specific exceptions like NoSuchElementException.

Hands-on Exercises (During the Session)

- **Exercise 1:** Use Implicit Wait to handle a slow-loading login form. (Solution)
- **Exercise 2:** Implement Explicit Wait for a dynamically loaded button. (Solution)
- **Exercise 3:** Use Fluent Wait to check for an element that appears randomly. (Solution)

Best Practices & Interview Questions

Best Practices:

- Prefer **Explicit Wait** over **Implicit Wait** for better control.
- Use **Fluent Wait** when dealing with AJAX elements.
- Avoid Thread.sleep() as it slows down execution.
- Optimize timeout values—**don't overuse high timeouts**.
- Handle **TimeoutException** properly.

Interview Questions:

- ❓ What is the difference between **Implicit, Explicit, and Fluent Wait**?
- ❓ What are the drawbacks of Implicit wait?
- ❓ When would you prefer **Fluent Wait over Explicit Wait**?
- ❓ What are the **common exceptions** while using waits?
- ❓ Can we use **both Implicit and Explicit Wait together**?
- ❓ How do you handle **AJAX-based dynamic elements**?

Post-Classroom Assignment (2 Hours)

Task 1: Automate login for an e-commerce site using Explicit Wait. (Solution)
Task 2: Use Fluent Wait to handle a dynamically appearing CAPTCHA. (Solution)
Task 3: Implement a Selenium script that waits for multiple elements before proceeding. (Solution)
Task 4: Optimize an existing script by replacing Thread.sleep() with Explicit Wait. (Solution)

Expected Outcomes by End of Day 2

- ✔ Clear understanding of **Implicit, Explicit, and Fluent Waits**.
- ✔ Ability to handle **slow-loading elements & dynamic web pages**.
- ✔ Hands-on experience with **different wait strategies**.
- ✔ Confidence to answer **client interview questions on waits**.
- ✔ Improved **test automation efficiency** using the best waiting mechanisms.

Solutions

✅ **Exercise 1:** Use Implicit Wait to handle a slow-loading login form.

Implicit Waits are used to set a default waiting time for the WebDriver to wait for an element to appear before throwing a `NoSuchElementException`.

Here's how you can use Implicit Wait:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import java.util.concurrent.TimeUnit;

public class ImplicitWaitExample {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Set implicit wait
```

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

        // Open the login page
        driver.get("https://example.com/login");

        username and password fields and login button
        WebElement usernameField = driver.findElement(By.id("username"));
        WebElement passwordField = driver.findElement(By.id("password"));
        WebElement loginButton = driver.findElement(By.id("loginButton"));

        // Perform login
        usernameField.sendKeys("testuser");
        passwordField.sendKeys("password");
        loginButton.click();

        // Close the browser
        driver.quit();
    }
}
```

✅ **Exercise 2:** Implement Explicit Wait for a dynamically loaded button.

Explicit Waits are used to wait for a specific condition to occur before proceeding further in the code.

Here's how you can use Explicit Wait:

```
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 ExplicitWaitExample {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the page with the dynamically loaded button
        driver.get("https://example.com/dynamicButton");

        // Set explicit wait
        WebDriverWait wait = new WebDriverWait(driver, 10);

        // Wait for the button to be clickable
        WebElement dynamicButton =
wait.until(ExpectedConditions.elementToBeClickable(By.id("dynamicButton")))
;

        // Click the button
        dynamicButton.click();
```

```
        // Close the browser
        driver.quit();
    }
}
```

✅ **Exercise 3:** Use Fluent Wait to check for an element that appears randomly.

Fluent Waits are used to define the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition.

Here's how you can use Fluent Wait:

```
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.FluentWait;
import java.time.Duration;
import java.util.NoSuchElementException;
import java.util.function.Function;

public class FluentWaitExample {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the page with the randomly appearing element
        driver.get("https://example.com/randomElement");

        // Set fluent wait
        FluentWait<WebDriver> wait = new FluentWait<>(driver)
                .withTimeout(Duration.ofSeconds(30))
                .pollingEvery(Duration.ofSeconds(5))
                .ignoring(NoSuchElementException.class);

        // Wait for the element to appear
        WebElement randomElement = wait.until(new Function<WebDriver,
WebElement>() {
            public WebElement apply(WebDriver driver) {
                return driver.findElement(By.id("randomElement"));
            }
        });

        // Perform an action with the element
        randomElement.click();

        // Close the browser
        driver.quit();
    }
}
```

Task 1: Automate login for an e-commerce site using Explicit Wait

Explicit Waits are used to wait for a specific condition to occur before proceeding further in the code. Here's an example of how to automate the login functionality of an e-commerce site using Explicit Wait:

```
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 ECommerceLoginAutomation {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the e-commerce website login page
        driver.get("https://example-ecommerce.com/login");

        // Set explicit wait
        WebDriverWait wait = new WebDriverWait(driver, 10);

        // Locate the username field and enter the username
        WebElement usernameField =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("username"))
);
        usernameField.sendKeys("testuser");

        // Locate the password field and enter the password
        WebElement passwordField =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("password"))
);
        passwordField.sendKeys("password");

        // Locate the login button and click it
        WebElement loginButton =
wait.until(ExpectedConditions.elementToBeClickable(By.id("loginButton")));
        loginButton.click();

        // Verify login by checking the presence of a logout button or user
profile element
        WebElement logoutButton =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("logoutButto
n")));
        if (logoutButton.isDisplayed()) {
            System.out.println("Login successful!");
        } else {
            System.out.println("Login failed.");
        }

        // Close the browser
        driver.quit();
    }
```

```
}
```

Task 2: Use Fluent Wait to handle a dynamically appearing CAPTCHA

Fluent Waits are used to define the maximum amount of time to wait for a condition, as well as the frequency with which to check the condition. Here's an example of how to handle a dynamically appearing CAPTCHA using Fluent Wait:

```
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.FluentWait;
import java.time.Duration;
import java.util.NoSuchElementException;
import java.util.function.Function;

public class FluentWaitCaptcha {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the page with the CAPTCHA
        driver.get("https://example.com/captcha");

        // Set fluent wait
        FluentWait<WebDriver> wait = new FluentWait<>(driver)
                .withTimeout(Duration.ofSeconds(30))
                .pollingEvery(Duration.ofSeconds(5))
                .ignoring(NoSuchElementException.class);

        // Wait for the CAPTCHA to appear
        WebElement captcha = wait.until(new Function<WebDriver,
WebElement>() {
            public WebElement apply(WebDriver driver) {
                return driver.findElement(By.id("captcha"));
            }
        });

        // Perform an action with the CAPTCHA (e.g., solve it)
        captcha.sendKeys("solvedCaptcha");

        // Close the browser
        driver.quit();
    }
}
```

Task 3: Implement a Selenium script that waits for multiple elements before proceeding

Here's an example of how to implement a Selenium script that waits for multiple elements before proceeding:

```
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 WaitForMultipleElements {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Set explicit wait
        WebDriverWait wait = new WebDriverWait(driver, 10);

        // Wait for multiple elements
        WebElement element1 =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("element1"))
);
        WebElement element2 =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("element2"))
);
        WebElement element3 =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("element3"))
);

        // Perform actions with the elements
        element1.click();
        element2.sendKeys("text");
        element3.click();

        // Close the browser
        driver.quit();
    }
}
```

Task 4: Optimize an existing script by replacing Thread.sleep() with Explicit Wait

Here's an example of how to optimize an existing script by replacing `Thread.sleep()` with Explicit Wait:

Original Script with `Thread.sleep()`:

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class OriginalScript {
```

```
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Perform an action (e.g., click a button)
        WebElement button = driver.findElement(By.id("button"));
        button.click();

        // Wait for the next element to load
        try {
            Thread.sleep(5000); // Replace this with Explicit Wait
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        // Perform an action with the next element
        WebElement nextElement = driver.findElement(By.id("nextElement"));
        nextElement.click();

        // Close the browser
        driver.quit();
    }
}
```

Optimized Script with Explicit Wait:

```
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;

publicizedScript {
    public static void main(String[] args) {
        // Set the path to the chromedriver executable
        System.setProperty("webdriver.chrome.driver",
"path/to/chromedriver");

        // Initialize WebDriver
        WebDriver driver = new ChromeDriver();

        // Open the webpage
        driver.get("https://example.com");

        // Perform an action (e.g., click a button)
        WebElement button = driver.findElement(By.id("button"));
        button.click();

        // Set explicit wait
        WebDriverWait wait = new WebDriverWait(driver, 10);

        // Wait for the next element to load
```

```
        WebElement nextElement =
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("nextElement
")));

        // Perform an action with the next element
        nextElement.click();

        // Close the browser
        driver.quit();
    }
}
```