In Selenium WebDriver, WebDriverWait is a powerful tool used to explicitly wait for certain conditions to occur before proceeding with the next step in your test script. This can be helpful when dealing with dynamic content or elements that take time to load.

Here's a breakdown of how to use WebDriverWait along with the different waiting conditions provided in Selenium:

1. Basic Syntax for WebDriverWait:

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

Here, driver is your WebDriver instance, and Duration.ofSeconds(10) is the time that WebDriver will wait before throwing a TimeoutException if the condition is not met.

Note: Starting from Selenium 4.x, the constructor has changed slightly, and you now use Duration instead of int for timeout.

2. Common Conditions Using ExpectedConditions

Selenium provides a class called ExpectedConditions to check various conditions, such as element visibility, element clicks, etc. Below are common methods used with WebDriverWait:

a. elementToBeClickable:

Waits until the element is both present in the DOM and clickable.

```
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("buttonId")));
element.click();
```

b. visibilityOfElementLocated:

Waits until the element is visible (i.e., the element exists in the DOM and is not hidden).

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

c. presenceOfElementLocated:

Waits until the element is present in the DOM (it may be hidden, but it exists).

```
WebElement element =
wait.until(ExpectedConditions.presenceOfElementLocated(By.id("elementId")));
```

d. visibilityOf:

Waits until the element is visible (it must be both in the DOM and visible to the user).

```
WebElement element = wait.until(ExpectedConditions.visibilityOf(someElement));
```

e. textToBePresentInElement:

Waits for the given text to be present in the element.

```
wait.until(ExpectedConditions.textToBePresentInElement(By.id("elementId"),"Expected
Text"));
```

f. alertIsPresent:

Waits until an alert is present.

```
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
alert.accept();
```

g. frameToBeAvailableAndSwitchToIt:

Waits until a frame is available and switches to it.

```
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.id("frameId")));
```

h. invisibilityOfElementLocated:

Waits for the element to become invisible.

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

i. stalenessOf:

Waits until an element is no longer attached to the DOM (useful when an element is refreshed or removed from the page).

```
wait.until(ExpectedConditions.stalenessOf(someElement));
```

j. elementToBeSelected:

Waits until the element is selected (useful for checkboxes, radio buttons).

```
WebElement element =
wait.until(ExpectedConditions.elementToBeSelected(By.id("checkboxId")));
```

k. numberOfElementsToBeMoreThan:

Waits until the number of elements in a list exceeds the given number.

```
wait.until(ExpectedConditions.numberOfElementsToBeMoreThan(By.className("className"),
5));
```

l. visibilityOfAllElements:

Waits until all elements in the list are visible.

```
List<WebElement> elements =
wait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(By.className("className")));
```

m. elementSelectionStateToBe:

Waits until the element's selection state (e.g., checkbox) matches the expected state (selected or unselected).

```
wait.until(ExpectedConditions.elementSelectionStateToBe(By.id("checkboxId"), true));  // true
for selected
```

3. Using Lambda Expression for Custom Conditions

With Selenium 4.x, you can also use WebDriverWait with lambda expressions to create custom conditions:

```
wait.until(driver -> driver.findElement(By.id("elementId")).isDisplayed());
```

This allows you to implement your own logic for waiting.

Example of Complete Code

Here’s an example of using WebDriverWait to wait for an element to be clickable, and then click it:

```
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;
import java.time.Duration;

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

// Set up WebDriverWait with a 10-second timeout
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

// Wait until the element is clickable
WebElement button =
wait.until(ExpectedConditions.elementToBeClickable(By.id("submitBtn")));

// Click the button after it becomes clickable
button.click();

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

Key Takeaways:

WebDriverWait is essential for waiting until a specific condition is met.

Use ExpectedConditions to specify the conditions (e.g., visibility, clickability).

Use Duration for timeout instead of an integer, which was the case in older versions of Selenium.

Lambda expressions can be used to implement custom conditions in modern Selenium versions.

4. Additional ExpectedConditions Methods:
a. elementToBeSelected:

Waits for the element to be selected (useful for checkboxes, radio buttons, or dropdowns).

```
WebElement checkbox =
wait.until(ExpectedConditions.elementToBeSelected(By.id("checkboxId")));
System.out.println("Checkbox is selected: " + checkbox.isSelected());
```

b. attributeToBe:

Waits for an element's attribute to match a specified value.

```
wait.until(ExpectedConditions.attributeToBe(
By.id("inputField"), "value", "Expected Value"));
```

This is useful when you want to wait until a text field or input element contains a certain value.

c. attributeContains:

Waits until the element's attribute contains the specified value (not necessarily an exact match).

```
wait.until(ExpectedConditions.attributeContains(
By.id("inputField"), "class", "active"));
```

This can be useful for checking if a class is added dynamically or for verifying partial attributes.

d. visibilityOfAllElementsLocatedBy:

Waits for all elements to be visible (often used for lists of elements).

```
List<WebElement> elements =
wait.until(ExpectedConditions.visibilityOfAllElementsLocatedBy(By.className("item")));
for (WebElement element : elements) {
System.out.println("Visible element: " + element.getText());
}
```

e. textToBePresentInElementLocated:

Waits until the text is present in the element.

```
wait.until(ExpectedConditions.textToBePresentInElementLocated(
By.id("status"), "Success"));
```

This is often used to verify that a page has been updated after an action (e.g., form submission).

f. invisibilityOfElementWithText:

Waits for an element to become invisible and for the text in that element to be a specific value.

```
wait.until(ExpectedConditions.invisibilityOfElementWithText(
By.id("loadingIndicator"), "Loading"));
```

This is useful to wait for elements to disappear after a certain condition has been met (like a "loading" indicator).

g. frameToBeAvailableAndSwitchToIt:

Waits until the frame is available and switches to it.

```
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(By.name("frameName")));
```

If you're working with pages containing multiple <iframe> elements, you might need to switch to the correct frame before interacting with elements inside it.

h. frameToBeAvailableAndSwitchToIt(by Index):

```
wait.until(ExpectedConditions.frameToBeAvailableAndSwitchToIt(0)); // Index of the frame
```

You can also wait for frames by index if the frame doesn't have a name or ID.

i. newWindowIsOpen:

Waits for a new window to open (e.g., when a link opens a new tab).

```
wait.until(ExpectedConditions.numberOfWindowsToBe(2)); // Waits until there are two
```

windows open

This is particularly useful in tests where the application opens a new window or tab.

5. Combining Multiple Conditions:

You can combine multiple conditions with WebDriverWait by using ExpectedConditions in combination with logical operators (like && or ||). For example, you might want to wait for an element to become visible and clickable.

```
wait.until(ExpectedConditions.and(
ExpectedConditions.visibilityOfElementLocated(By.id("elementId")),
ExpectedConditions.elementToBeClickable(By.id("elementId"))
));
```

This will ensure that the element is both visible and clickable before proceeding.

6. Custom Conditions:

You can define custom wait conditions by using lambda expressions with WebDriverWait. For example, waiting for a specific element's text to change dynamically:

```
wait.until(driver -> driver.findElement(By.id("status")).getText().equals("Success"));
```

This custom condition allows you to wait for specific text content in an element, which is common when testing dynamic content changes (e.g., after a form submission).

Alternatively, you can implement a custom condition that checks the status of a page or element beyond what's available in ExpectedConditions.

7. Waiting for JavaScript Alerts:

You can also use WebDriverWait to wait for JavaScript alerts to appear before interacting with them. This is useful if you're testing pop-ups or alerts on your web page.

```
Alert alert = wait.until(ExpectedConditions.alertIsPresent());
alert.accept();
```

This ensures that your test will only continue once the alert is present and ready for interaction.

8. Waiting for Element to Have CSS Value:

Sometimes, you need to wait until an element's CSS style changes. For instance, you can wait for an element to be displayed (change from display: none to display: block):

```
wait.until(driver -> driver.findElement(By.id("elementId"))
.getCssValue("display").equals("block"));
```

This waits for the element to be visible by checking its CSS value for display.

9. Waiting for Condition in a Loop:

You can implement a custom loop to repeatedly check for a condition, retrying until it succeeds or the timeout occurs.

Example: Waiting for an element to be visible in a loop:

```
boolean isVisible = false;
long startTime = System.currentTimeMillis();
while (System.currentTimeMillis() - startTime < 10000) { // 10-second timeout
try {
WebElement element = driver.findElement(By.id("elementId"));
if (element.isDisplayed()) {
isVisible = true;
break;
}
} catch (NoSuchElementException e) {
// Handle exception (element not found)
}
Thread.sleep(500);  // Wait a short period before retrying
}
if (!isVisible) {
throw new TimeoutException("Element was not visible after 10 seconds");
}
```

This can be useful for custom waits where you might need to handle retries in a specific way.

10. Polling Interval:

Starting from Selenium 4, you can specify a polling interval. This defines how often WebDriver will check the condition before returning or throwing a timeout exception.

```
WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10),
Duration.ofMillis(500));
wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementId")));
```

Here, the condition is checked every 500 milliseconds until the timeout is reached.

11. Best Practices:

Use Explicit Waits Over Implicit Waits: Prefer WebDriverWait (explicit waits) over implicit waits, as explicit waits are more flexible and specific to certain conditions.

Avoid Hard-Coding Timings: Avoid using hard-coded Thread.sleep() delays, as they can make your tests brittle. Instead, rely on WebDriverWait to wait for specific conditions.

Test for Element State, Not Just Visibility: Often, waiting for visibility is not enough. Consider waiting for elements to be clickable, enabled, or present.

Limit Timeout Duration: Set appropriate timeouts. A very long wait can mask issues, while too short of a timeout might cause tests to fail prematurely.

Handle Exceptions Gracefully: Always handle exceptions like TimeoutException in your tests and log useful information about the failure.

Example: Combining Multiple Conditions

Let's look at an example where we wait for an element to be clickable, visible, and contain a specific text:

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

// Wait until the element is clickable
WebElement element =
wait.until(ExpectedConditions.elementToBeClickable(By.id("submitBtn")));

// Wait until the element is visible
wait.until(ExpectedConditions.visibilityOf(element));
```

```
// Wait until the element's text contains the expected value
wait.until(ExpectedConditions.textToBePresentInElement(element,"Submit"));
```

This combines several conditions in one place, making the code more robust and readable.