Handling StaleElementReferenceException in Selenium

Handling StaleElementReferenceException in Selenium occurs when an element that was found earlier in the DOM is no longer valid or present. This often happens if the DOM is refreshed or updated. Here are five different ways to handle this exception using Selenium WebDriverManager in Java, along with real-time examples.

1. Retrying to Find the Element

Retrying to find the element after catching the exception ensures that we are working with the most current element in the DOM.

```java
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;


public class HandleStaleElement {

    public static void main(String[] args) {

        WebDriverManager.chromedriver().setup();

        WebDriver driver = new ChromeDriver();

        driver.get("http://example.com");


        By elementLocator = By.id("dynamicElement");


        for (int i = 0; i < 3; i++) {

            try {
                WebElement element = driver.findElement(elementLocator);

                element.click();

                break;

            } catch (StaleElementReferenceException e) {

                // Retry to find the element

            }

        }


        driver.quit();

    }

}
```

2. Using Fluent Wait

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

```
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.FluentWait;

import org.openqa.selenium.support.ui.Wait;

import io.github.bonigarcia.wdm.WebDriverManager;


import java.time.Duration;
```

```
public class HandleStaleElement {

    public static void main(String[] args) {

        WebDriverManager.chromedriver().setup();

        WebDriver driver = new ChromeDriver();

        driver.get("http://example.com");


        By elementLocator = By.id("dynamicElement");


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

                .withTimeout(Duration.ofSeconds(30))

                .pollingEvery(Duration.ofSeconds(5))

                .ignoring(StaleElementReferenceException.class);


                                                              WebElement       element       =
wait.until(ExpectedConditions.elementToBeClickable(elementLocator));

        element.click();


        driver.quit();

    }

}
```

3. Using Try-Catch with a Loop

Using a loop to continuously try to interact with the element until it succeeds or the maximum attempts are reached.

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;
```

```java
import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;



public class HandleStaleElement {

    public static void main(String[] args) {

        WebDriverManager.chromedriver().setup();

        WebDriver driver = new ChromeDriver();

        driver.get("http://example.com");



        By elementLocator = By.id("dynamicElement");



        int attempts = 0;

        while (attempts < 5) {

            try {

                WebElement element = driver.findElement(elementLocator);

                element.click();

                break;

            } catch (StaleElementReferenceException e) {

                attempts++;

            }

        }



        driver.quit();

    }

}
```

4. Refetching the Element

Refetching the element inside the catch block to ensure it references the most current version of the element in the DOM.

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.WebElement;

import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;


public class HandleStaleElement {

    public static void main(String[] args) {

        WebDriverManager.chromedriver().setup();

        WebDriver driver = new ChromeDriver();

        driver.get("http://example.com");


        By elementLocator = By.id("dynamicElement");


        WebElement element = driver.findElement(elementLocator);

        try {

            element.click();

        } catch (StaleElementReferenceException e) {

            element = driver.findElement(elementLocator); // Refetch the element

            element.click();

        }


        driver.quit();
    }
}
```

5. Waiting for the Element to be Refreshed

Waiting for a specific condition that indicates the element has been refreshed in the DOM.

```
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 io.github.bonigarcia.wdm.WebDriverManager;


public class HandleStaleElement {
    public static void main(String[] args) {
        WebDriverManager.chromedriver().setup();
        WebDriver driver = new ChromeDriver();
        driver.get("http://example.com");


        By elementLocator = By.id("dynamicElement");


        WebDriverWait wait = new WebDriverWait(driver, 10);
                                                              WebElement       element       =
wait.until(ExpectedConditions.elementToBeClickable(elementLocator));


        try {
            element.click();
        } catch (StaleElementReferenceException e) {
                                                                                element       =
wait.until(ExpectedConditions.refreshed(ExpectedConditions.elementToBeClickable(elementL
ocator)));
            element.click();
        }

        driver.quit();
    }
}
```

Real-time Example

Suppose you have a dynamic web page where an element's state can change upon user interaction or through AJAX calls. For instance, a web page with a 'Load More' button that fetches more items to the list, and each item has a 'Details' button. Handling StaleElementReferenceException is crucial as the DOM might change frequently. Using the methods above ensures your script can handle these dynamic changes effectively.