Exercise: Selenium Waits

0. Prerequisites

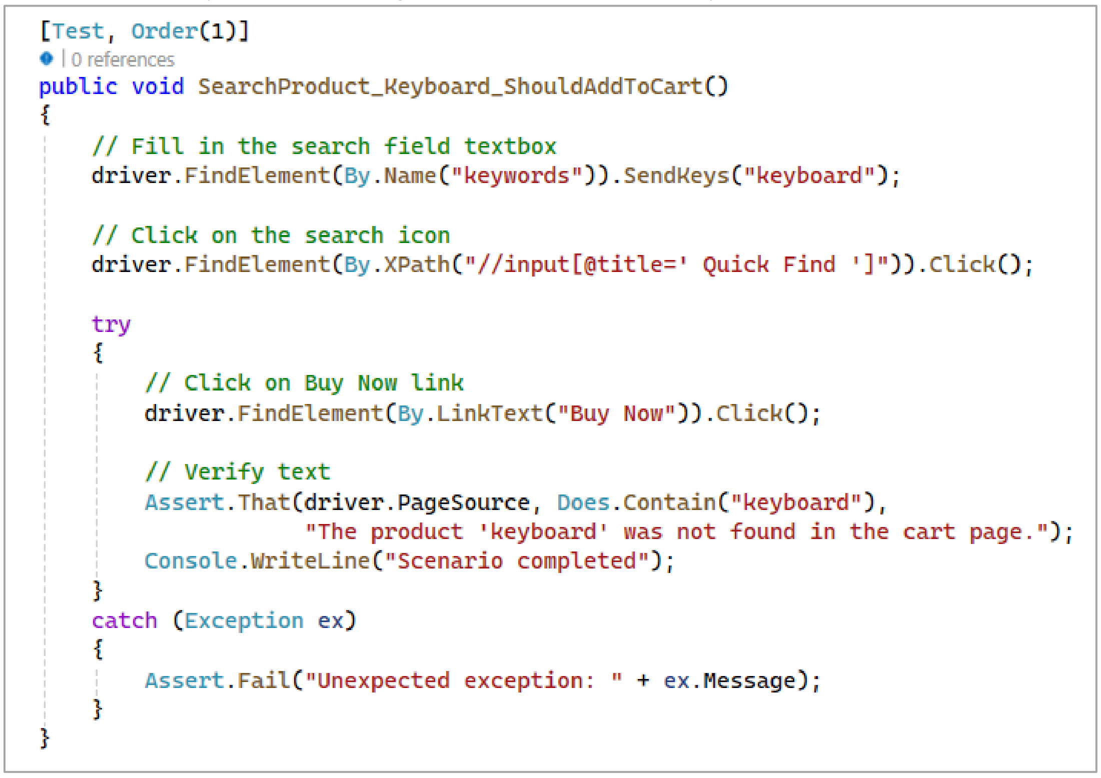
For each of the exercises create a separate NUnit project with the following:

- Install the necessary Selenium packages via NuGet:
  Selenium.WebDriver, Selenium.WebDriver.ChromeDriver, Selenium.Support.
- Initialize the ChromeDriver and navigate to the application URL.
- Create a new class for your Selenium tests.
- Include necessary namespaces for Selenium and NUnit.
- Define a test fixture class with a setup method to initialize the WebDriver.
- Define a tear down method.

1. Search Product with Implicit Wait

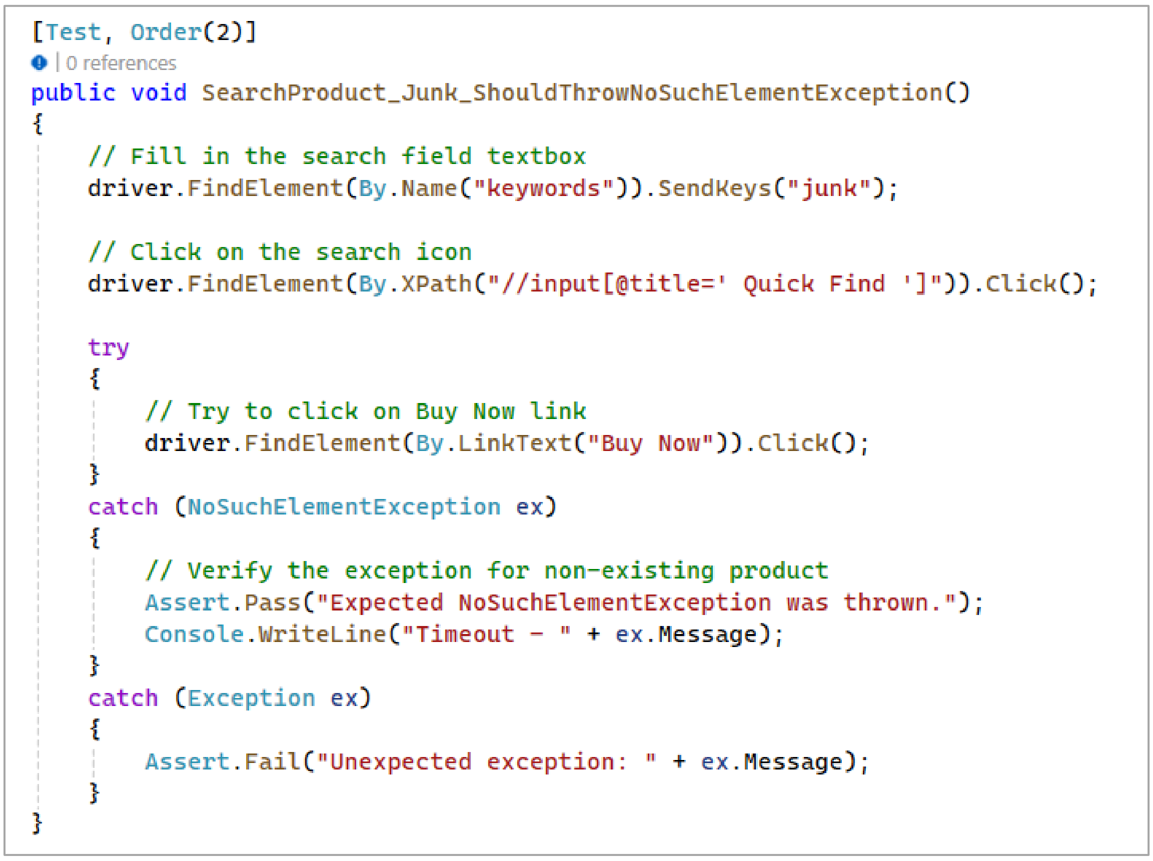
- Application URL: http://practice.bpbonline.com/
- In the setup method, configure the WebDriver to use an implicit wait:
  driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);

- **Create a Test: Search for "keyboard"**
  - Implement a test method to search for the product "keyboard".
  - Verify the result using assertions to ensure the product is found and added to the cart.



```
[Test, Order(1)]
● | 0 references
public void SearchProduct_Keyboard_ShouldAddToCart()
{
    // Fill in the search field textbox
    driver.FindElement(By.Name("keywords")).SendKeys("keyboard");

    // Click on the search icon
    driver.FindElement(By.XPath("//input[@title=' Quick Find ']")).Click();

    try
    {
        // Click on Buy Now link
        driver.FindElement(By.LinkText("Buy Now")).Click();

        // Verify text
        Assert.That(driver.PageSource, Does.Contain("keyboard"),
                    "The product 'keyboard' was not found in the cart page.");
        Console.WriteLine("Scenario completed");
    }
    catch (Exception ex)
    {
        Assert.Fail("Unexpected exception: " + ex.Message);
    }
}
```

- **Create the Second Test: Search for "junk"**
  - Implement a test method to search for a non-existing product "junk".
  - The implicit wait will still apply, but since the product doesn't exist, the test should handle the No Such Element Exception.
  - Use assertions to verify that the correct exception is thrown.



```
[Test, Order(2)]
0 references
public void SearchProduct_Junk_ShouldThrowNoSuchElementException()
{
    // Fill in the search field textbox
    driver.FindElement(By.Name("keywords")).SendKeys("junk");

    // Click on the search icon
    driver.FindElement(By.XPath("//input[@title=' Quick Find ']")).Click();

    try
    {
        // Try to click on Buy Now link
        driver.FindElement(By.LinkText("Buy Now")).Click();
    }
    catch (NoSuchElementException ex)
    {
        // Verify the exception for non-existing product
        Assert.Pass("Expected NoSuchElementException was thrown.");
        Console.WriteLine("Timeout - " + ex.Message);
    }
    catch (Exception ex)
    {
        Assert.Fail("Unexpected exception: " + ex.Message);
    }
}
```

- **Run the tests.**
    - For the "keyboard" product, it should be found and added to the cart.
    - For the "junk" product, a No Such Element Exception should be thrown due to the non-existence of the product.

Implicit Waits Explained:
The implicit wait is set once in the setup method and applies to all elements throughout the test.
It helps in synchronizing the test by waiting up to a specified time for elements to appear.

2. Search Product with Explicit Wait

- Application URL: http://practice.bpbonline.com/
- Define a test fixture class with a setup method to initialize the WebDriver.
- **In the setup method, configure the WebDriver to set an initial implicit wait, which will be adjusted later for explicit waits:**
  driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);

Note: The initial implicit wait ensures that the driver will wait up to 10 seconds for elements to appear during the initial setup phase.

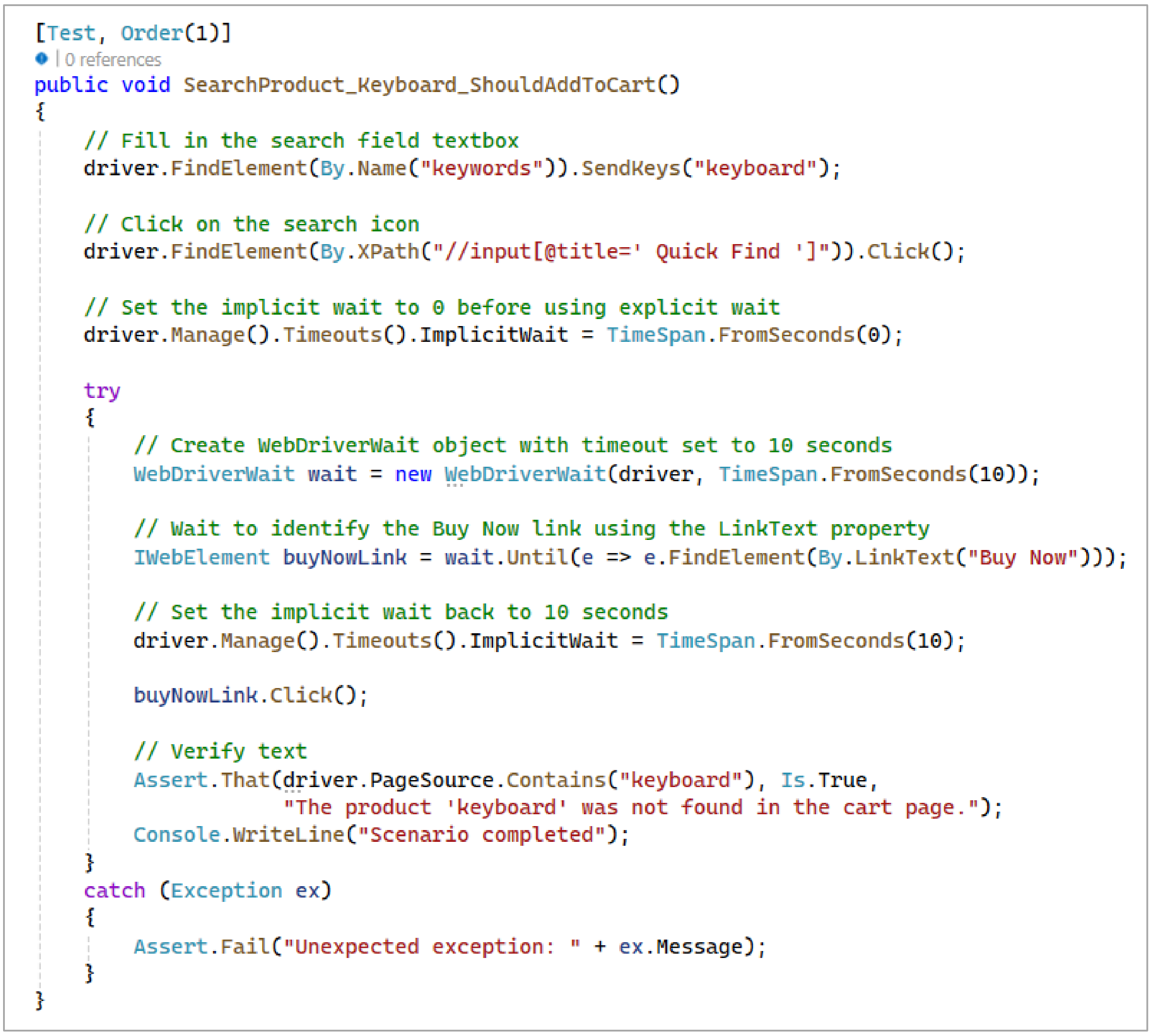
- **Create the First Test: Search for "keyboard"**
    - Implement a test method to search for the product "keyboard".
    - After performing initial actions, set the implicit wait to zero before using explicit wait:
      driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(0);

Note: Setting the implicit wait to zero before using explicit wait ensures that the implicit wait does not interfere with the explicit wait. This gives precise control over specific elements' wait times.

- **Use WebDriverWait to wait for the "Buy Now" button to appear:**
  WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

Note: Explicit wait is used here to wait for up to 10 seconds for the "Buy Now" button to appear, allowing the test to handle dynamic content effectively.

- Use assertions to verify that the product is found and can be added to the cart.

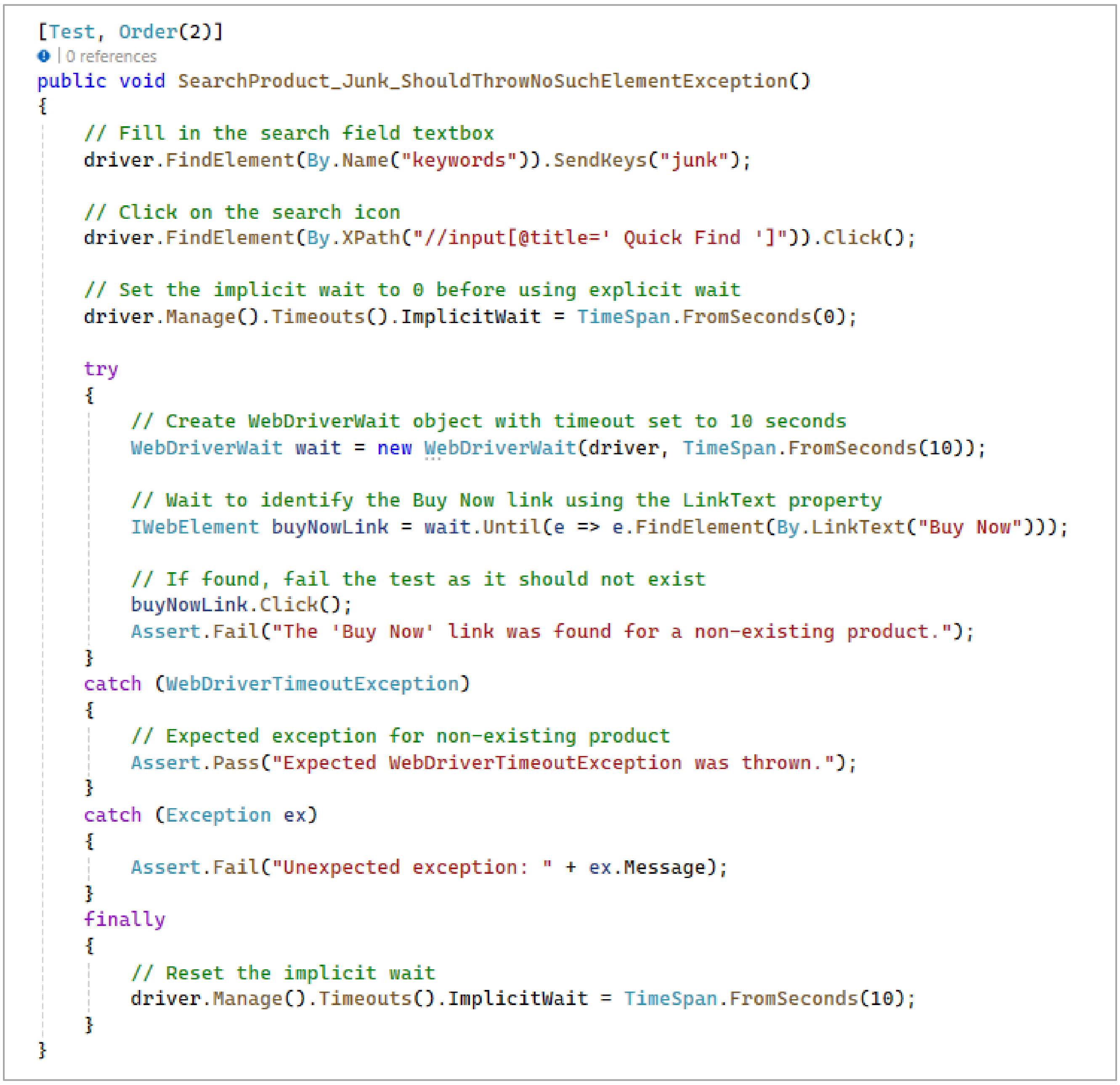


```
[Test, Order(1)]
● | 0 references
public void SearchProduct_Keyboard_ShouldAddToCart()
{
    // Fill in the search field textbox
    driver.FindElement(By.Name("keywords")).SendKeys("keyboard");

    // Click on the search icon
    driver.FindElement(By.XPath("//input[@title=' Quick Find ']")).Click();

    // Set the implicit wait to 0 before using explicit wait
    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(0);

    try
    {
        // Create WebDriverWait object with timeout set to 10 seconds
        WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

        // Wait to identify the Buy Now link using the LinkText property
        IWebElement buyNowLink = wait.Until(e => e.FindElement(By.LinkText("Buy Now")));

        // Set the implicit wait back to 10 seconds
        driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);

        buyNowLink.Click();

        // Verify text
        Assert.That(driver.PageSource.Contains("keyboard"), Is.True,
                    "The product 'keyboard' was not found in the cart page.");
        Console.WriteLine("Scenario completed");
    }
    catch (Exception ex)
    {
        Assert.Fail("Unexpected exception: " + ex.Message);
    }
}
```

- **Create the Second Test: Search for "junk"**
    - Implement a test method to search for a non-existing product "junk".
    - Similarly, set the implicit wait to zero before using explicit wait.
    - Use WebDriverWait to wait for the "Buy Now" button, which will not appear, leading to a TimeoutException.

Note: By setting the implicit wait to zero and using an explicit wait, we can specifically wait for the "Buy Now" button and handle the situation where it does not appear within the specified time.

- Use assertions to verify that the correct exception is thrown.



```
[Test, Order(2)]
0 references
public void SearchProduct_Junk_ShouldThrowNoSuchElementException()
{
    // Fill in the search field textbox
    driver.FindElement(By.Name("keywords")).SendKeys("junk");

    // Click on the search icon
    driver.FindElement(By.XPath("//input[@title=' Quick Find ']")).Click();

    // Set the implicit wait to 0 before using explicit wait
    driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(0);

    try
    {
        // Create WebDriverWait object with timeout set to 10 seconds
        WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10));

        // Wait to identify the Buy Now link using the LinkText property
        IWebElement buyNowLink = wait.Until(e => e.FindElement(By.LinkText("Buy Now")));

        // If found, fail the test as it should not exist
        buyNowLink.Click();
        Assert.Fail("The 'Buy Now' link was found for a non-existing product.");
    }
    catch (WebDriverTimeoutException)
    {
        // Expected exception for non-existing product
        Assert.Pass("Expected WebDriverTimeoutException was thrown.");
    }
    catch (Exception ex)
    {
        Assert.Fail("Unexpected exception: " + ex.Message);
    }
    finally
    {
        // Reset the implicit wait
        driver.Manage().Timeouts().ImplicitWait = TimeSpan.FromSeconds(10);
    }
}
```

- **Run the tests.**
  - For the "keyboard" product, it should be found and added to the cart.
  - For the "junk" product, a TimeoutException should be thrown due to the non-existence of the product.

Explicit Waits Explained:
Explicit waits are used for specific conditions and elements, providing better control over synchronization issues. By setting the implicit wait to zero, we avoid interference with the explicit wait, allowing for precise timing control. It helps in handling scenarios where different elements require different wait times. Useful for dynamic content where certain elements may take varying amounts of time to appear.

3. Working with Windows

This exercise will guide you through the steps to handle multiple browser windows in Selenium. We will demonstrate switching between windows, interacting with them, and handling scenarios where a window might not exist.

- Application URL: https://the-internet.herokuapp.com/windows

- In the setup method, configure the WebDriver to use an implicit wait.

Understanding Window Handles:

- Every browser window opened by Selenium is identified by a unique window handle.
- Use **`driver.WindowHandles`** to retrieve all the window handles currently opened by the web driver.

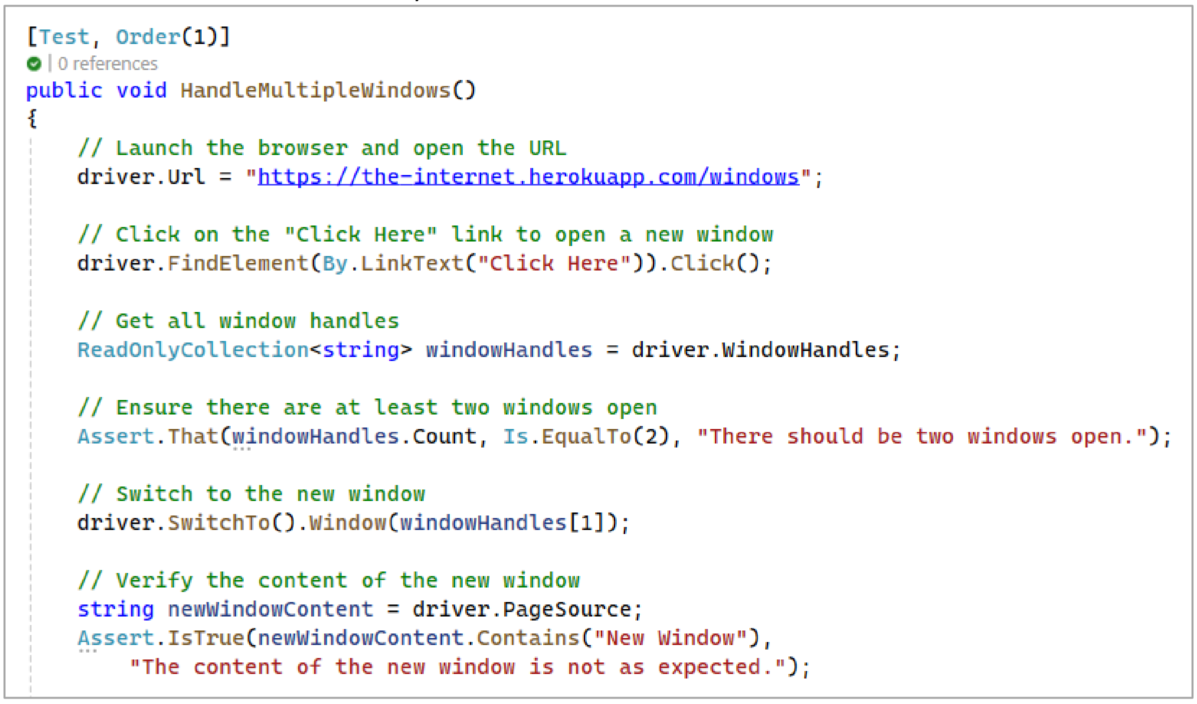
Switching Between Windows:

- Use **`driver.SwitchTo().Window(windowHandle)`** to switch the context to the specified window. This allows you to perform actions and validations within that window.
- It's crucial to keep track of the window handles if your test involves multiple windows, to ensure you switch back to the correct window after performing operations on a child window.

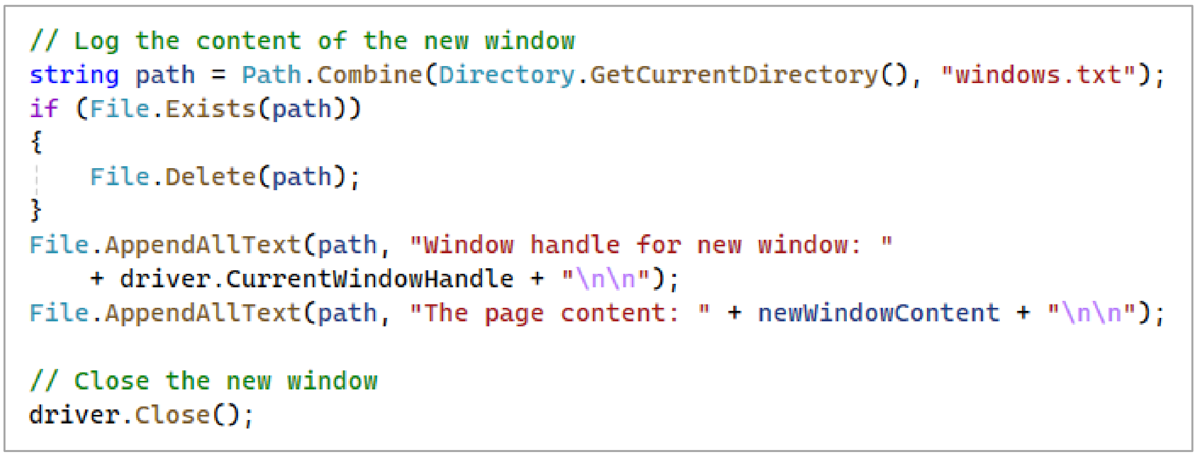
Managing Multiple Windows:

- Always ensure to close the child windows once done to avoid resource leaks.
- Use assertions to verify that you are indeed interacting with the correct window by checking the content or title of the window.

- **Create the First Test: Handling Multiple Windows**
  - Implement a test method to interact with multiple windows.
  - Use WindowHandles to get all window IDs.
  - Switch between windows using SwitchTo().Window().
  - Use assertions to verify the interactions and content of each window.



```csharp
[Test, Order(1)]
| 0 references
public void HandleMultipleWindows()
{
    // Launch the browser and open the URL
    driver.Url = "https://the-internet.herokuapp.com/windows";

    // Click on the "Click Here" link to open a new window
    driver.FindElement(By.LinkText("Click Here")).Click();

    // Get all window handles
    ReadOnlyCollection<string> windowHandles = driver.WindowHandles;

    // Ensure there are at least two windows open
    Assert.That(windowHandles.Count, Is.EqualTo(2), "There should be two windows open.");

    // Switch to the new window
    driver.SwitchTo().Window(windowHandles[1]);

    // Verify the content of the new window
    string newWindowContent = driver.PageSource;
    Assert.IsTrue(newWindowContent.Contains("New Window"),
        "The content of the new window is not as expected.");
```



```csharp
// Log the content of the new window
string path = Path.Combine(Directory.GetCurrentDirectory(), "windows.txt");
if (File.Exists(path))
{
    File.Delete(path);
}
File.AppendAllText(path, "Window handle for new window: "
    + driver.CurrentWindowHandle + "\n\n");
File.AppendAllText(path, "The page content: " + newWindowContent + "\n\n");

// Close the new window
driver.Close();
```

```
    // Switch back to the original window
    driver.SwitchTo().Window(windowHandles[0]);

    // Verify the content of the original window
    string originalWindowContent = driver.PageSource;
    Assert.That(originalWindowContent, Does.Contain("Opening a new window"),
                "The content of the original window is not as expected.");

    // Log the content of the original window
    File.AppendAllText(path, "Window handle for original window: "
        + driver.CurrentWindowHandle + "\n\n");
    File.AppendAllText(path, "The page content: " + originalWindowContent + "\n\n");
}
```

- **Create the Second Test: Handling NoSuchWindowException**
  - Implement a test method to handle scenarios where a window does not exist.
  - Attempt to switch to a closed window and catch the NoSuchWindowException.
  - Use assertions to verify the correct exception handling.

```
[Test, Order(2)]
| 0 references
public void HandleNoSuchWindowException()
{
    // Launch the browser and open the URL
    driver.Url = "https://the-internet.herokuapp.com/windows";

    // Click on the "Click Here" link to open a new window
    driver.FindElement(By.LinkText("Click Here")).Click();

    // Get all window handles
    ReadOnlyCollection<string> windowHandles = driver.WindowHandles;

    // Switch to the new window
    driver.SwitchTo().Window(windowHandles[1]);

    // Close the new window
    driver.Close();
```

```
    try
    {
        // Attempt to switch back to the closed window
        driver.SwitchTo().Window(windowHandles[1]);
    }
    catch (NoSuchWindowException ex)
    {
        // Log the exception
        string path = Path.Combine(Directory.GetCurrentDirectory(), "windows.txt");
        File.AppendAllText(path, "NoSuchWindowException caught: " + ex.Message + "\n\n");
        Assert.Pass("NoSuchWindowException was correctly handled.");
    }
    catch (Exception ex)
    {
        Assert.Fail("An unexpected exception was thrown: " + ex.Message);
    }
    finally
    {
        // Switch back to the original window
        driver.SwitchTo().Window(windowHandles[0]);
    }
}
```

- Run the tests.
  - For multiple windows, ensure correct navigation and interaction.
  - For closed windows, ensure correct exception handling.

4. Working with Alerts

This exercise will guide you through the steps to handle different types of alerts in Selenium and validate the outcomes using assertions. We will demonstrate handling basic JavaScript alerts, confirm alerts, and prompt alerts.

- Application URL: https://the-internet.herokuapp.com/javascript_alerts
- In the setup method, configure the WebDriver to use an implicit wait.

Understanding Alerts:

- Alerts are modal windows that require user interaction to proceed. They can be informational, confirmation, or prompt alerts.
- Informational alerts require the user to acknowledge the message by clicking OK.
- Confirmation alerts require the user to either accept (OK) or dismiss (Cancel) the alert.
- Prompt alerts require the user to enter text before accepting or dismissing the alert.

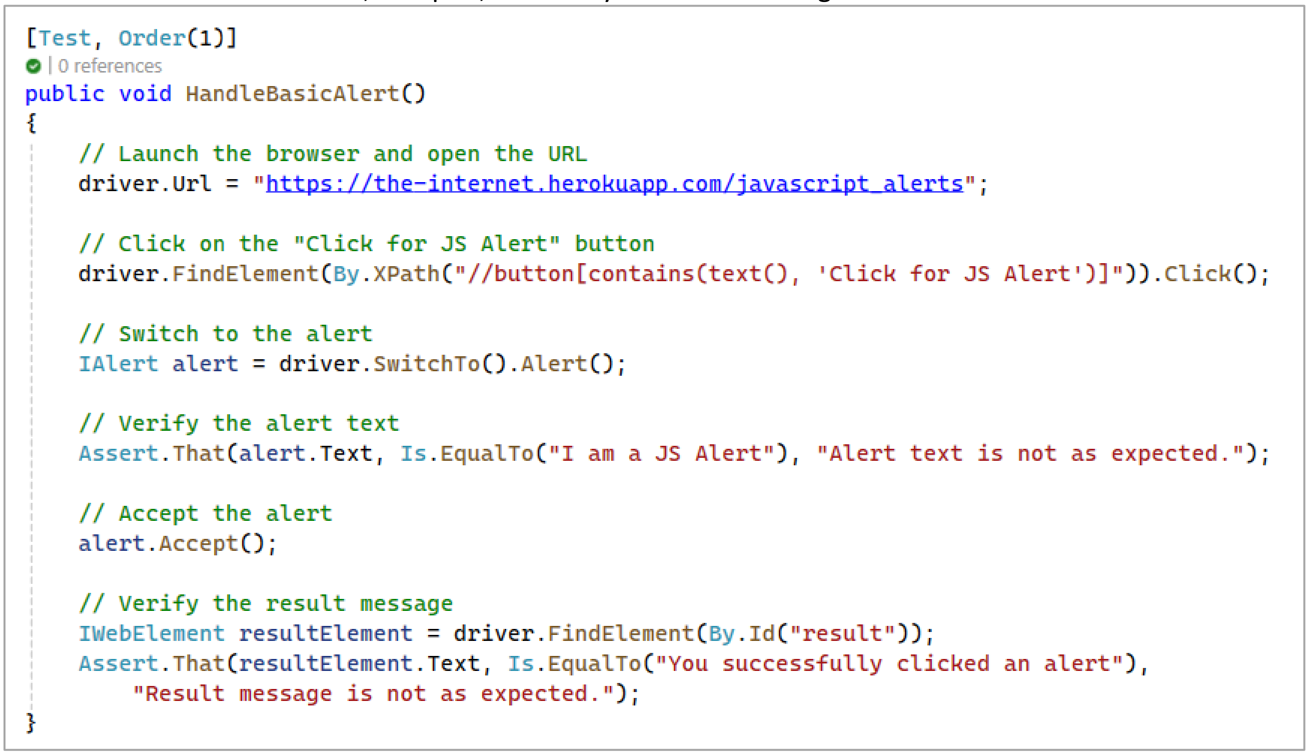
Handling Alerts:

- Selenium provides the **`IAlert`** interface to interact with alerts.
- Use **`driver.SwitchTo().Alert()`** to switch the context to the alert.
- The IAlert interface provides methods such as **`Accept()`**, **`Dismiss()`**, **`SendKeys()`**, and **`GetText()`** to interact with alerts.

Types of Alerts:

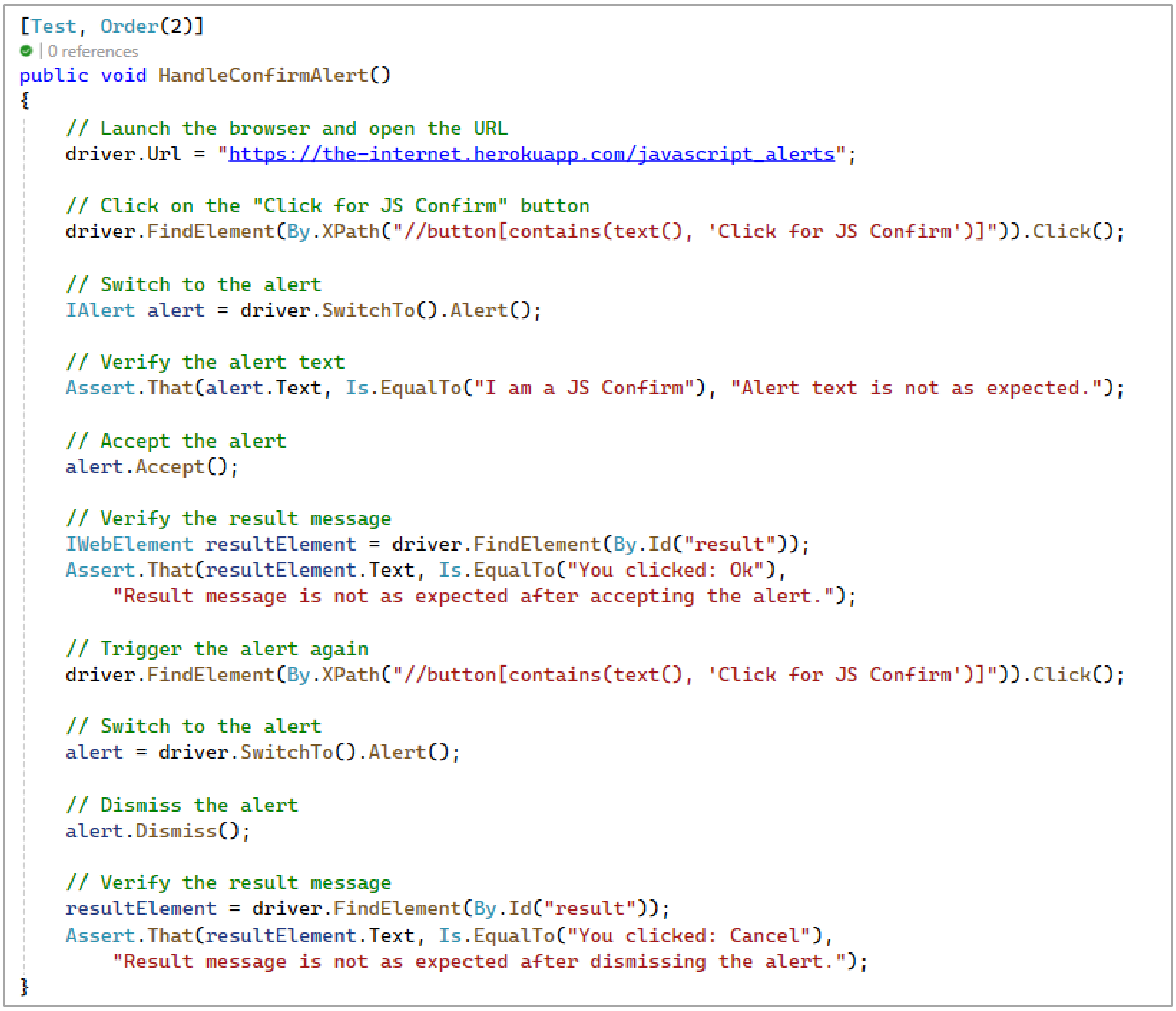
- **Basic Alerts:** These alerts only have an OK button. Use the **`Accept()`** method to close them.
- **Confirmation Alerts:** These alerts have OK and Cancel buttons. Use the **`Accept()`** method to click OK and the **`Dismiss()`** method to click Cancel.
- **Prompt Alerts:** These alerts require user input. Use the **`SendKeys()`** method to enter text, followed by the Accept() or Dismiss() methods.

- **Create the First Test: Handling Basic JavaScript Alerts**
  - Implement a test method to handle a basic JavaScript alert.
  - Click the button to trigger the alert.
  - Switch to the alert, accept it, and verify the result message.



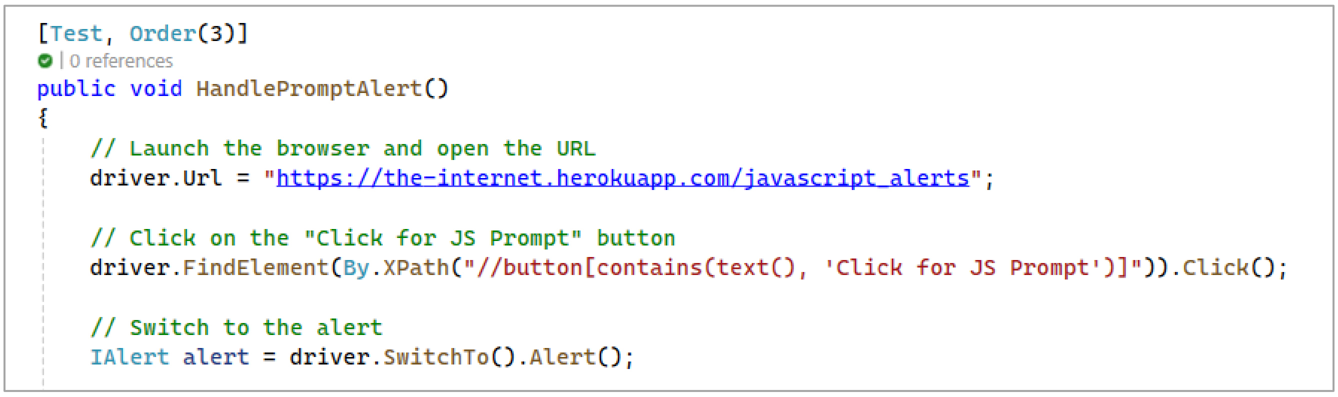
```
[Test, Order(1)]
| 0 references
public void HandleBasicAlert()
{
    // Launch the browser and open the URL
    driver.Url = "https://the-internet.herokuapp.com/javascript_alerts";

    // Click on the "Click for JS Alert" button
    driver.FindElement(By.XPath("//button[contains(text(), 'Click for JS Alert')]")).Click();

    // Switch to the alert
    IAlert alert = driver.SwitchTo().Alert();

    // Verify the alert text
    Assert.That(alert.Text, Is.EqualTo("I am a JS Alert"), "Alert text is not as expected.");

    // Accept the alert
    alert.Accept();

    // Verify the result message
    IWebElement resultElement = driver.FindElement(By.Id("result"));
    Assert.That(resultElement.Text, Is.EqualTo("You successfully clicked an alert"),
        "Result message is not as expected.");
}
```

- **Create the Second Test: Handling JavaScript Confirm Alerts**
    - Implement a test method to handle a JavaScript confirm alert.
    - Click the button to trigger the alert.
    - Switch to the alert, accept it, and verify the result message.
    - Trigger the alert again, dismiss it, and verify the result message.



```
[Test, Order(2)]
| 0 references
public void HandleConfirmAlert()
{
    // Launch the browser and open the URL
    driver.Url = "https://the-internet.herokuapp.com/javascript_alerts";

    // Click on the "Click for JS Confirm" button
    driver.FindElement(By.XPath("//button[contains(text(), 'Click for JS Confirm')]")).Click();

    // Switch to the alert
    IAlert alert = driver.SwitchTo().Alert();

    // Verify the alert text
    Assert.That(alert.Text, Is.EqualTo("I am a JS Confirm"), "Alert text is not as expected.");

    // Accept the alert
    alert.Accept();

    // Verify the result message
    IWebElement resultElement = driver.FindElement(By.Id("result"));
    Assert.That(resultElement.Text, Is.EqualTo("You clicked: Ok"),
        "Result message is not as expected after accepting the alert.");

    // Trigger the alert again
    driver.FindElement(By.XPath("//button[contains(text(), 'Click for JS Confirm')]")).Click();

    // Switch to the alert
    alert = driver.SwitchTo().Alert();

    // Dismiss the alert
    alert.Dismiss();

    // Verify the result message
    resultElement = driver.FindElement(By.Id("result"));
    Assert.That(resultElement.Text, Is.EqualTo("You clicked: Cancel"),
        "Result message is not as expected after dismissing the alert.");
}
```

- **Create the Third Test: Handling JavaScript Prompt Alerts**
    - Implement a test method to handle a JavaScript prompt alert.
    - Click the button to trigger the alert.
    - Switch to the alert, send text input, accept it, and verify the result message.
    - Tear Down the WebDriver



```
[Test, Order(3)]
| 0 references
public void HandlePromptAlert()
{
    // Launch the browser and open the URL
    driver.Url = "https://the-internet.herokuapp.com/javascript_alerts";

    // Click on the "Click for JS Prompt" button
    driver.FindElement(By.XPath("//button[contains(text(), 'Click for JS Prompt')]")).Click();

    // Switch to the alert
    IAlert alert = driver.SwitchTo().Alert();
    // Verify the alert text
    Assert.That(alert.Text, Is.EqualTo("I am a JS prompt"), "Alert text is not as expected.");

    // Send text to the alert
    string inputText = "Hello there!";
    alert.SendKeys(inputText);

    // Accept the alert
    alert.Accept();

    // Verify the result message
    IWebElement resultElement = driver.FindElement(By.Id("result"));
    Assert.That(resultElement.Text, Is.EqualTo("You entered: " + inputText),
        "Result message is not as expected after entering text in the prompt.");
}
```

- **Run the tests.**
    - Verify that the test results match the expected outcomes for each type of alert.

5. Working with iFrames

This exercise will guide you through the steps to handle iFrames in Selenium and validate the outcomes using assertions. We will demonstrate handling iFrames by switching to them using index, ID, and WebElement.

We'll be using this simple CodePen page where we have a dropdown button inside an iFrame.
Application URL: [CodePen iFrame Example]()
In the setup method, configure the WebDriver to use an implicit wait.

Understanding iFrames:

- iFrames are HTML elements that allow embedding external content within a parent HTML document, such as web pages or media.
- iFrames create isolated environments that can't be accessed directly using normal Selenium commands. This requires switching the context of the WebDriver to the iFrame to interact with its elements.
- To interact with elements inside iFrames, Selenium provides the switch_to.frame() method.

Handling iFrames:

- Use Selenium's switch_to.frame() method to switch the context to the desired iFrame.
- Selenium provides three approaches to select and switch to the desired frame: using a WebElement, specifying a name or ID, or indicating an index.

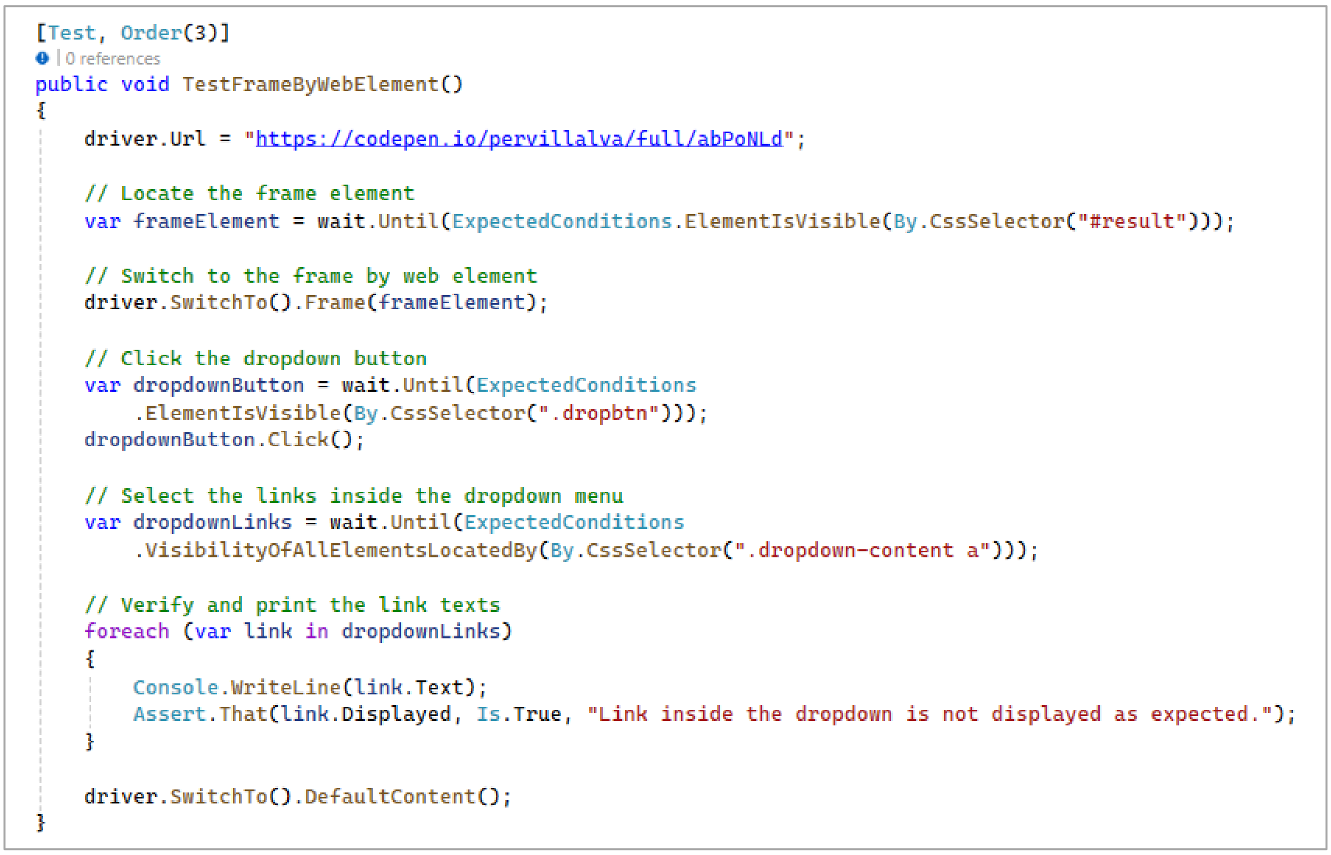
- **Create the First Test: Handling iFrames by Index**
    - Implement a test method to handle an iFrame by its index.
    - Navigate to the application URL.
    - Switch to the iFrame by index.
    - Click the dropdown button inside the iFrame.
    - Select the links inside the dropdown menu.
    - Verify and print the link texts.
    - Switch back to the default content.

```
[Test, Order(1)]
| 0 references
public void TestFrameByIndex()
{
    driver.Url = "https://codepen.io/pervillalva/full/abPoNLd";

    // Wait until the iframe is available and switch to it by finding the first iframe
    wait.Until(ExpectedConditions.FrameToBeAvailableAndSwitchToIt(By.TagName("iframe")));

    // Click the dropdown button
    var dropdownButton = wait.Until(ExpectedConditions
        .ElementIsVisible(By.CssSelector(".dropbtn")));
    dropdownButton.Click();

    // Select the links inside the dropdown menu
    var dropdownLinks = wait.Until(ExpectedConditions
        .VisibilityOfAllElementsLocatedBy(By.CssSelector(".dropdown-content a")));

    // Verify and print the link texts
    foreach (var link in dropdownLinks)
    {
        Console.WriteLine(link.Text);
        Assert.That(link.Displayed, Is.True, "Link inside the dropdown is not displayed as expected.");
    }

    driver.SwitchTo().DefaultContent();
}
```

- **Create the Second Test: Handling iFrames by ID**
  - Implement a test method to handle an iFrame by its ID.
  - Navigate to the application URL.
  - Switch to the iFrame by its ID.
  - Click the dropdown button inside the iFrame.
  - Select the links inside the dropdown menu.
  - Verify and print the link texts.
  - Switch back to the default content.

```
[Test, Order(2)]
| 0 references
public void TestFrameById()
{
    driver.Url = "https://codepen.io/pervillalva/full/abPoNLd";

    // Wait until the iframe is available and switch to it by ID
    wait.Until(ExpectedConditions.FrameToBeAvailableAndSwitchToIt("result"));

    // Click the dropdown button
    var dropdownButton = wait.Until(ExpectedConditions
        .ElementIsVisible(By.CssSelector(".dropbtn")));
    dropdownButton.Click();

    // Select the links inside the dropdown menu
    var dropdownLinks = wait.Until(ExpectedConditions
        .VisibilityOfAllElementsLocatedBy(By.CssSelector(".dropdown-content a")));

    // Verify and print the link texts
    foreach (var link in dropdownLinks)
    {
        Console.WriteLine(link.Text);
        Assert.That(link.Displayed, Is.True, "Link inside the dropdown is not displayed as expected.");
    }

    driver.SwitchTo().DefaultContent();
}
```

- **Create the Third Test: Handling iFrames by WebElement**
  - Implement a test method to handle an iFrame by its WebElement.
  - Navigate to the application URL.
  - Locate the iFrame using a CSS selector.
  - Switch to the iFrame by the located WebElement.
  - Click the dropdown button inside the iFrame.
  - Select the links inside the dropdown menu.
  - Verify and print the link texts.
  - Switch back to the default content.



```csharp
[Test, Order(3)]
| 0 references
public void TestFrameByWebElement()
{
    driver.Url = "https://codepen.io/pervillalva/full/abPoNLd";

    // Locate the frame element
    var frameElement = wait.Until(ExpectedConditions.ElementIsVisible(By.CssSelector("#result")));

    // Switch to the frame by web element
    driver.SwitchTo().Frame(frameElement);

    // Click the dropdown button
    var dropdownButton = wait.Until(ExpectedConditions
        .ElementIsVisible(By.CssSelector(".dropbtn")));
    dropdownButton.Click();

    // Select the links inside the dropdown menu
    var dropdownLinks = wait.Until(ExpectedConditions
        .VisibilityOfAllElementsLocatedBy(By.CssSelector(".dropdown-content a")));

    // Verify and print the link texts
    foreach (var link in dropdownLinks)
    {
        Console.WriteLine(link.Text);
        Assert.That(link.Displayed, Is.True, "Link inside the dropdown is not displayed as expected.");
    }

    driver.SwitchTo().DefaultContent();
}
```