Exception Handling

**What is an Exception?**

Exceptions are events due to which java program ends abruptly without giving expected output. Java provides a framework where a user can handle exceptions.

The process of handling Exceptions is called Exception Handling.

Exceptions need to be handled because they break the normal flow of execution of a program. One of the important intentions of exception handling is to prevent this break and continue program execution. Sometimes, you might want to perform some series of actions on occurring of a certain exception.

When an exception occurs, an exception object is created which is technically referred to as ‘*Throwing an Exception’* and we add *Try/Catch* blocks like,

|  |
| --- |
| **try** {  // Protected code  } **catch** (ExceptionName e) {  // Catch block  } |

**#1)** The piece of code which might throw an exception is added inside the Try block.

**#2)** The Catch statement catches the exception and takes it as a parameter.

**#3)** When no exception is thrown, the try statement is executed and not the catch statement.

**Example**: When selenium script fails due to the wrong locator, then the developer should be able to understand the reason for failure and this can be achieved easily if the exception is handled properly in the program.

In my experience, it is best to try to avoid WebDriver exceptions whenever possible and catch truly exceptional cases. Use try/catch to handle things that go wrong and are outside my control.

*Avoid the ones I can Catch others!*

This is the best strategy that has worked for me.

**For example,** consider a test page that takes more than usual time to load on a test server. We will get frequent exceptions while doing actions on this page. So, instead of just catching this every time, we can

* Add a wait command and try to avoid an exception
* Use ‘Try/Catch’ to handle in case if a truly exceptional case has occurred

Thereby reducing the chances for exceptions.

Frequent exceptions in Selenium

* Webdriver
* Timeout
* Nosuchelementexception
* Nosuchwindow
* Noalertpresent

### Avoiding And Handling Common Exceptions

***Let’s discuss Avoid-And-Handle approach for the above-mentioned exceptions:***

#### **#1) org.openqa.selenium.NoSuchElementException**

This commonly seen exception class is a subclass of **NotFoundException**class. The exception occurs when WebDriver is unable to find and locate elements.

Usually, this happens when tester writes incorrect element locator in the findElement(By, by) method.

Consider that in the below example, correct id for the text field was ‘firstfield’ but the tester incorrectly mentioned it as ‘fistfield’. In this case, WebDriver cannot locate the element and org.openqa.selenium.NoSuchElementException will be thrown

|  |
| --- |
| driver.findElement(By.id("submit")).click();  Exception Handling:    **try** {  driver.findElement(By.id("submit")).click();  } **catch** (NoSuchElementException e)  {  System.out.println(“no such element”);  } |

In this case, the exception is thrown even if the element is not loaded.

**Avoiding-And-Handling**: Try giving a wait command.

***Example:*** The wait command below waits 10 seconds for the presence of web element with id ‘submit’. Then it tries to click it. If the element is available but still the click fails, an exception is caught.

Using delayed time is a common practice in test automation to create a pause in between the steps. By adding a Try/Catch we ensure that the program continues even if the wait couldn’t help.

|  |
| --- |
| **try** {  WebDriverWait wait = **new** WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.presenceOfElementLocated(By.id("submit")));  **try** {  driver.findElement(By.id("submit")).click();  } **catch** (WebDriverException e) {  System.out.println(“An exceptional **case**.”);  }  } **catch** (TimeOutException e) {  System.out.println(“WebDriver couldn’t locate the element”);  } |

#### **#2) org.openqa.selenium.NoSuchWindowException**

NoSuchWindowException comes under NotFoundException class. This is thrown when WebDriver tries to switch to an invalid window.

The below code can throw org.openqa.selenium.NoSuchWindowException if the window handle doesn’t exist or is not available to switch.

driver.switchTo().window(handle\_1);

**Avoiding-And-Handling**: We would use window handles to get the set of active windows and then perform actions on the same.

In the example below, for each window handle, driver switch to is executed. Therefore chances of passing a wrong window parameter reduced.

|  |
| --- |
| **for** (String handle : driver.getWindowHandles()) {  **try** {  driver.switchTo().window(handle);  } **catch** (NoSuchWindowException e) {  System.out.println(“An exceptional **case**”);  }  } |

#### **#3) org.openqa.selenium.NoSuchFrameException**

When WebDriver is trying to switch to an invalid frame, NoSuchFrameException under NotFoundException class is thrown.

The below code can throw org.openqa.selenium.NoSuchFrameException if a frame “frame\_11” doesn’t exist or is not available.

driver.switchTo().frame(“frame\_11”);

Exception Handling:

|  |
| --- |
| **try** {  driver.switchTo().frame("frame\_11");  } **catch** (NoSuchFrameException e) |

In this case, the exception is thrown even if the frame is not loaded.

**Avoiding-And-Handling**: Try to give a wait command.

In the example below, WebDriver waits for 10 seconds for the frame to be available. If the frame is available and still there is an exception, then it is caught.

|  |
| --- |
| **try** {  WebDriverWait wait = **new** WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.frameToBeAvaliableAndSwitchToIt(frame\_11));  **try** {  driver.switchTo().frame("frame\_11");  } **catch** (WebDriverException e) {  System.out.println(“An exceptional **case**”);  }  } **catch** (NoSuchFrameException e) {  System.out.println(“WebDriver couldn’t locate the frame”);  } |

#### **#4) org.openqa.selenium.NoAlertPresentException**

**NoAlertPresentException under NotFoundException** is thrown when WebDriver tries to switch to an alert, which is not available.

org.openqa.selenium.NoAlertPresentException will be thrown If below automation code calls accept() operation on Alert() class when an alert is not yet on the screen.

driver.switchTo().alert().accept();

Exception Handling:

|  |
| --- |
| **try** {  driver.switchTo().alert().accept();  } **catch** (NoSuchAlertException e)  {  } |

In this case, the exception is thrown even if the alert is not loaded completely.

**Avoiding-And-Handling**: Always use explicit or fluent wait for a particular time in all cases where an alert is expected. If the alert is available and still there is an exception, then it is caught.

|  |
| --- |
| **try** {  WebDriverWait wait = **new** WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.alertIsPresent());  **try** {  driver.switchTo().alert().accept();  } **catch** (NoAlertPresentException e) {  System.out.println(“An exceptional **case**”);  }  } **catch** (TimeOutException e)  System.out.println(“WebDriver couldn’t locate the Alert”);  } |

#### **#5) org.openqa.selenium.InvalidSelectorException**

This subclass of NoSuchElementException class occurs when a selector is incorrect or syntactically invalid. This exception occurs commonly when XPATH locator is used.

**Consider the below example:**

clickXPathButtonAndWait(“//button[@type=’button’][100]”);

This would throw an InvalidSelectorExeption because the XPATH syntax is incorrect.

**Avoiding and Handling**: To avoid this, we should check the locator used because the locator is likely incorrect or the syntax is wrong. Using Firebug to find xpath can reduce this exception.

**Below code shows how to handle it using Try/Catch**

|  |
| --- |
| **try** {  clickXPathButtonAndWait("//button[@type='button']");  } **catch** (InvalidSelectorException e) {  } |

#### **#6) org.openqa.selenium.ElementNotVisibleException**

ElementNotVisibleException class is a subclass of ElementNotInteractableException class. This exception is thrown when WebDriver tries to perform an action on an invisible web element, which cannot be interacted with. That is, the web element is in a hidden state.

**For example,** in the below code, if the type of button with id ‘submit’ is ‘hidden’ in HTML, org.openqa.selenium.ElementNotVisibleException will be thrown.

|  |
| --- |
| driver.findElement(By.id("submit")).click();  Exception Handling:  **try** {  driver.findElement(By.id("submit")).click();  } **catch** (ElementNotVisibleException e) |

In this case, the exception is thrown even if the page has not loaded completely.

**Avoiding-And-Handling**: There are two ways to do this. We can either use wait for the element to get completely.

The below code waits 10 seconds for the element. If the element is visible and still exception is thrown, it is caught.

|  |
| --- |
| **try** {  WebDriverWait wait = **new** WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions.visibilityOfElementLocated(By.id(”submit”));  **try** {  driver.findElement(By.id("submit")).click();  } **catch** (WebDriverException e) {  System.out.println(“Exceptional **case**”);  }  } **catch** (TimeOutException e)  System.out.println(“WebDriver couldn’t find **this** element visible”);  } |

#### **#7) org.openqa.selenium.ElementNotSelectableException**

This exception comes under InvalidElementStateException class. ElementNotSelectableException indicates that the web element is present in the web page but cannot be selected.

**For example,** the below code can throw a ElementNotSelectableException if the id “swift” is disabled.

Select dropdown = new Select(driver.findElement(By.id(“swift”)));

**Exception Handling:**

try {  
Select dropdown = new Select(driver.findElement(By.id(“swift”)));  
} catch (ElementNotSelectableException e)

In this case, exception is thrown even if the element becomes enabled after a while.

**Avoiding-And-Handling**: We can add a wait command to wait until the element becomes clickable. If there is still an exception, it is caught.

|  |
| --- |
| **try** {  WebDriverWait wait = **new** WebDriverWait(driver, TimeSpan.FromSeconds(10));  wait.Until(ExpectedConditions. elementToBeClickable(By.id(”swift”));  **try** {  Select dropdown = **new** Select(driver.findElement(By.id("swift")));  } **catch** (WebDriverException e) {  System.out.println(“Exceptional **case**”);  }  } **catch** (TimeOutException e)  System.out.println(“WebDriver found that **this** element was not selectable.”);  } |

#### **#8) org.openqa.selenium.TimeoutException**

This exception occurs when a command completion takes more than the wait time. Waits are mainly used in WebDriver to avoid the exception ElementNotVisibleException.

Sometimes test page might not load completely before next command in the program. If WebDriver tries to find an element in the webpage before the page completely loads, then exception ElementNotVisibleException is thrown. To avoid this exception, waits commands are added.

However, if the components don’t load even after the wait, the exception org.openqa.selenium**.TimeoutException**will be thrown.

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

driver.get(“https://www.softwaretestinghelp.com” );

In the above program, an implicit wait of 10 seconds is added. If the page www.softwaretestinghelp.com doesn’t load in 10 seconds, then TimeoutException will be thrown.

**Avoiding and Handling**: To avoid this, we can manually check the average time for a page to load and adjust the wait

Or, we can add explicit wait using JavaScript executor until the page is loaded.

In the below example, JavaScript executor is used. After page navigation, we call JavaScript return document.readyState for 20 seconds until “complete” is returned.

|  |
| --- |
| WebDriverWait wait = **new** WebDriverWait(driver, TimeSpan.FromSeconds(30));    wait.until(webDriver -> ((JavascriptExecutor)webDriver).executeScript("return document.readyState").equals("complete"));    driver.get("https://www.softwaretestinghelp.com"); |

#### **#9) org.openqa.selenium.NoSuchSessionException**

This exception is thrown when a method is called after quitting the browser by WebDriver.quit(). This can also happen due to web browser issues like crashes and WebDriver cannot execute any command using the driver instance.

To see this exception, the code below can be executed.

driver.quit()

Select dropdown = new Select(driver.findElement(By.id(“swift”)));

**Avoiding and Handling**: Always choose the latest stable version of browser to run Selenium Webdriver testcases.

This exception can be reduced by using driver.quit() at the completion of all tests. Do not try to use them after each test case. This can lead to issues when driver instance is null and upcoming test cases try to use it without initializing.

The below code creates WebDriver instance in the @BeforeSuite TestiNG annotation and destroys it in @AfterSuite TestiNG annotation

|  |
| --- |
| @BeforeSuite  **public** **void** setUp() **throws** MalformedURLException {  WebDriver driver = **new** FirefoxDriver();  }  @AfterSuite  **public** **void** testDown() {  driver.quit();  } |

#### **#10) org.openqa.selenium.StaleElementReferenceException**

This exception says that a web element is no longer present in the web page.

This error is not the same as ElementNotVisibleException.

StaleElementReferenceException is thrown when an object for a particular web element was created in the program without any problem and however; this element is no longer present in the window. This can happen if there was a

* Navigation to another page
* DOM has refreshed
* A frame or window switch

WebElement firstName = driver.findElement(By.id(“firstname”));

driver.switchTo().window(Child\_Window);

element.sendKeys(“Aaron”);

In the code above, object firstName was created and then the window was switched. Then, WebDriver tries to type ‘Aaron’ in the form field. In this case StaleElementReferenceException is thrown.

**Avoiding and Handling**: Confirm that we are trying to do the action in the correct window. To avoid issues due to DOM refresh, we can use Dynamic Xpath

Let’s discuss another example.

Say ‘id’ of a username field is ‘username\_1’ and the XPath will be //\*[@id=’firstname\_1?]. When you open the page again the ‘id’ might change say to ‘’firstname \_11’. In this case, the test will fail because the WebDriver could not find the element. In this case, StaleElementReferenceException will be thrown.

In this case, we can use a dynamic xpath like,

|  |
| --- |
| **try** {  driver.findElement(By.xpath(“//\*[contains(@id,firstname’)]”)).sendKeys(“Aaron”);  } **catch** (StaleElementReferenceException e) |

In the example above dynamic XPATH is used and if the exception is still found, it is caught.

### Conclusion

Exception handling is the essential part of every java program as well as selenium script. We can build robust and optimal code by **handling an exception in smart ways**. And it is also a best practice to handle exceptions in a script which will give you a better report when a program fails due to any reason.

Here we have tried to cover the process and framework of exception handling which is required to be implemented in selenium scripts.

Remember it is not mandatory to always handle the exception in a try-catch block. You can also throw an exception depending upon the requirement in a script.

An exception shouldn’t be ignored as they break program execution. In this tutorial, we went through different exceptions and ways to reduce the chances of getting them through manual checks and codes.

Adding waits can control some cases like ‘NoSuchElementException‘, ‘ElementNotFoundException‘, ‘ElementNotVisibleException‘.

If we do not use exception handling.

A screenshot of a computer

Description automatically generated

In real time we should use try , catch

Throwable is a class . not a keyword

(Exception e) – here Exception is the parent class.

Filenotfount – parent is IO exception- parent is exception