When you work with Selenium you willencounter many exceptions. Solving of those exceptions would be sometimes very tricky but If you read the exception the answer to solving is in the meaning of Exception name. We will be going through some most commonly used Exception in Selenium.

* NoSuchAttributeException find\_element\_by\_\* can’t find the element.
* NoSuchElementException find\_element\_by\_\* can’t find the element.
* InvalidSelectorException Thrown when the selector which is used to find an element does not return a WebElement. Currently, this only happens when the selector is an XPath expression is used which is either syntactically invalid (i.e. it is not an XPath expression) or the expression does not select WebElements.
* ElementNotVisibleException Thrown to indicate that although an element is present on the DOM, it is not visible, and so is not able to be interacted with.
* RemoteDriverServerException
* NoSuchWindowException
* ElementNotSelectableException
* NoSuchFrameException
* WebDriverException
* UnexpectedTagNameException Thrown when a support class did not get an expected web element
* UnableToSetCookieException Thrown when a driver fails to set a cookie.
* MoveTargetOutOfBoundsException Indicates that the target provided to the actions move() method is invalid.
* InvalidSwitchToTargetException The frame or window target to be switched doesn’t exist.
* InvalidElementStateException
* InvalidCookieDomainException Thrown when attempting to add a cookie under a different domain than the current URL
* ImeNotAvailableException Indicates that IME support is not available. This exception is thrown for every IME-related method call if IME support is not available on the machine.
* ImeActivationFailedException Indicates that activating an IME engine has failed.
* ErrorInResponseException An error has occurred on the server side.
* TimeoutException Thrown when a command does not complete in enough time.
* StaleElementReferenceException Indicates that a reference to an element is now “stale” — the element no longer appears on the DOM of the page.

**Top 10 Selenium Exceptions and How To Handle These (Exact Code)**

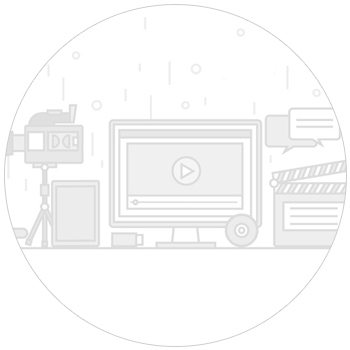
Last Updated:[November 10, 2019](https://www.softwaretestinghelp.com/exception-handling-framework-selenium-tutorial-19/)

**Handling Selenium WebDriver Exceptions Using Exception Handling Framework – Selenium Tutorial #19**

Getting an exception in automation code is very common. ‘Exception’ as the word indicates, is a special or uncommon case.

Automation code execution usually may not carry out as expected due to many factors involved in execution such as network stability issues, Internet issues, server stability etc. We might get exceptions due to insufficient waiting time or incorrect syntaxes, parameters etc.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Exception-handling-in-Selenium-2.jpg)



In the last WebDriver tutorial, we learned about 3 different types of important web elements like [Web Tables, Frames and Dynamic elements](https://www.softwaretestinghelp.com/selenium-tutorial-18/) and their handling mechanisms in selenium script

Before moving ahead with Framework tutorials in this [Selenium training](https://www.softwaretestinghelp.com/selenium-tutorial-1/) series, here in this tutorial we will learn about **types of exceptions and how to handle exceptions in Java and Selenium scripts**. Developers/testers use exception handling framework to handle an exception in selenium scripts.

What You Will Learn: [[show](https://www.softwaretestinghelp.com/exception-handling-framework-selenium-tutorial-19/)]

**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,

|  |  |
| --- | --- |
| 1 | try { |
| 2 | // Protected code | |

|  |  |  |
| --- | --- | --- |
| 3 | } catch (ExceptionName e) { | |
| 4 | // Catch block |

|  |  |
| --- | --- |
| 5 | } |

**#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.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/05/Exceptions-in-selenium-webdriver.jpg)

**Advantages and Disadvantages of Avoid-Handle approach**

| **Advantages** | **Disadvantages** |
| --- | --- |
| 1) This approach reduces the chances of getting exceptions. | 1) Increases the lines of codes because you add extra code to avoid exceptions |
| If an exception is still caught it 2) would be a truly exceptional case which is worth checking | 2) Should have a better understanding of Web Driver API, commands and exceptions |
| Reduce debugging time. Automation code is intended to find bugs and you don’t want to see too many unwanted 3) exceptions and find the reasons behind each of them |  |
| 4) In the Catch block, you deal with more valid cases |  |
| 5) Reduce false failures |  |
| 6) Clearer report |  |

In this tutorial, we will discuss *Avoid-And-Handle* approach for the 10 most common exceptions in Selenium WebDriver. Before that, let's get a basic understanding of Exception Handling and Try/Catch blocks.

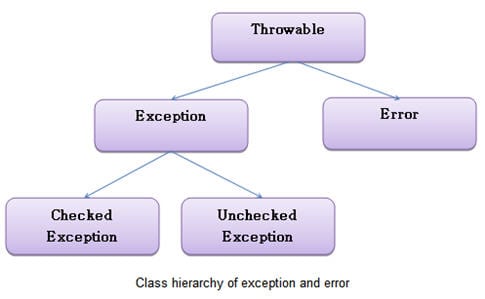
**Types of Exceptions in Java and Selenium**

Below we have described the types of exceptions and the different ways how we can use exception handling framework in selenium scripts.

**There are three kinds of exceptions:**

1. Checked Exception
2. Unchecked Exception
3. Error

**The class hierarchy of exception and error:**

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2014/11/Exception-handling-in-Selenium.jpg)

**#1) Checked Exception:** Checked exception is handled during compile time and it gives the compilation error if it is not caught and handled during compile time.

**Example**: *FileNotFoundException*, *IOException* etc.

**#2) Unchecked Exception:** In case of the unchecked exception, a compiler does not mandate to handle. The compiler ignores during compile time.

**Example**: *ArrayIndexoutOfBoundException*

**#3) Error:** When a scenario is fatal and the program cannot recover then JVM throws an error. Errors cannot be handled by the try-catch block. Even if the user tries to handle the error by using Try catch block, it cannot recover from the error.

**Example**: *Assertion error*, *OutOfMemoryError* etc.

**Exception Handling**

**Try and Catch block:**

*try-catch* blocks are generally used to handle exceptions. Type of exceptions is declared in catch block which is expected to come. When an exception comes in try block, immediately control moves to catch block.

**Example**:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | br = new BufferedReader(new FileReader("Data")); | |

|  |  |  |
| --- | --- | --- |
| 3 | } catch(IOException ie) | |
| 4 | { |

|  |  |  |
| --- | --- | --- |
| 5 | ie.printStackTrace(); | |
| 6 | } |

There can be multiple catch blocks for one try block depending upon the type of exception.

**Example**:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | br = new BufferedReader(new FileReader("Data")); | |

|  |  |  |
| --- | --- | --- |
| 3 | } catch(IOException ie) | |
| 4 | { |

|  |  |
| --- | --- |
| 5 | ie.printStackTrace(); |
| 6 | } catch(FileNotFoundException file){ | |

|  |  |  |
| --- | --- | --- |
| 7 | file.printStackTrace(); | |
| 8 | } |

***throws* Exception:**

*throws* keyword in java is used to throw an exception rather than handling it. All checked exceptions can be thrown by methods.

**Example**:

|  |  |  |
| --- | --- | --- |
| 1 | public static void main(String[] args) throws IOException | |
| 2 | { |

|  |  |  |
| --- | --- | --- |
| 3 | BufferedReader br=new BufferedReader(new FileReader("Data")); | |
| 4 | while ((line = br.readLine()) != null) |

|  |  |
| --- | --- |
| 5 | { |
| 6 | System.out.println(line); | |

|  |  |  |
| --- | --- | --- |
| 7 | } | |
| 8 | } |

***finally* block:**

*finally*, block executes irrespective of execution of try-catch block and it executes immediately after try/catch block completes.

Basically file close, database connection etc. can be closed in finally block.

**Example**:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | br = new BufferedReader(new FileReader("Data")); | |

|  |  |  |
| --- | --- | --- |
| 3 | } catch(IOException ie) | |
| 4 | { |

|  |  |  |
| --- | --- | --- |
| 5 | ie.printStackTrace(); | |
| 6 | } |

|  |  |
| --- | --- |
| 7 | Finally { |
| 8 | br.close(); | |

|  |  |
| --- | --- |
| 9 | } |

In the above example, *BufferReader* stream is closed in finally block. *br.close()* will always execute irrespective of execution of try and catch block.

***Note***: finally block can exist without any catch block. It is not necessary to have a catch block always.

There can be many catch blocks but only one finally block can be used.

***Throwable***: Throwable is a parent class for error and exception. Generally, it is difficult to handle errors in java. If a programmer is not sure about the type of error and exception, then it is advised to use the Throwable class which can catch both error and exception.

**Example**:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | br = new BufferedReader(new FileReader("Data")); | |

|  |  |  |
| --- | --- | --- |
| 3 | } catch (Throwable t) | |
| 4 | { |

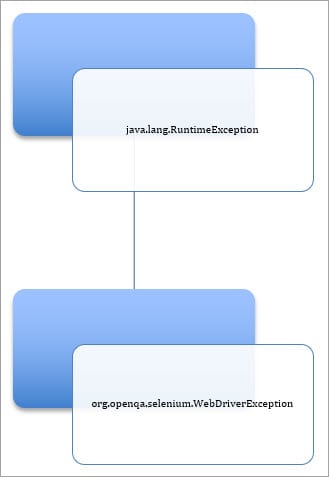
|  |  |  |
| --- | --- | --- |
| 5 | t.printStackTrace(); | |
| 6 | } |

**Common Exceptions in Selenium WebDriver**

Selenium has its own set of exceptions. While developing selenium scripts, a programmer has to handle or throw those exceptions.

**Below are a few examples of exceptions in selenium:**

All runtime exception classes in Selenium WebDriver come under the superclass WebDriverException.

[](https://cdn.softwaretestinghelp.com/wp-content/qa/uploads/2018/05/Webdriver-exceptions.jpg)

Though there are many Exception classes under WebDriverException, we commonly see the below ones.

* *NoSuchElementException*
* *NoSuchWindowException*
* *NoSuchFrameException*
* *NoAlertPresentException*
* *InvalidSelectorException*
* *ElementNotVisibleException*
* *ElementNotSelectableException*
* *TimeoutException*
* *NoSuchSessionException*
* *StaleElementReferenceException*

**Details**:

***ElementNotVisibleException***: If selenium tries to find an element but the element is not visible within the page

***NoAlertPresentException***: If a user tries to handle an alert box but the alert is not present.

***NoSuchAttributeException***: While trying to get attribute value but the attribute is not available in DOM.

***NoSuchElementException***: This exception is due to accessing an element which is not available on the page.

***WebDriverException***: Exception comes when a code is unable to initialize WebDriver.

**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

|  |  |  |
| --- | --- | --- |
| 1 | driver.findElement(By.id("submit")).click(); | |
| 2 | Exception Handling: |

|  |  |
| --- | --- |
| 3 |  |
| 4 | try { | |

|  |  |  |
| --- | --- | --- |
| 5 | driver.findElement(By.id("submit")).click(); | |
| 6 | } catch (NoSuchElementException e) |

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.

|  |  |
| --- | --- |
| 1 | try { |
| 2 | WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10)); | |

|  |  |  |
| --- | --- | --- |
| 3 | wait.Until(ExpectedConditions.presenceOfElementLocated(By.id("submit"))); | |
| 4 | try { |

|  |  |  |
| --- | --- | --- |
| 5 | driver.findElement(By.id("submit")).click(); | |
| 6 | } catch (WebDriverException e) { |

|  |  |  |
| --- | --- | --- |
| 7 | System.out.println(“An exceptional case.”); | |
| 8 | } |

|  |  |  |
| --- | --- | --- |
| 9 | } catch (TimeOutException e) { | |
| 10 | | System.out.println(“WebDriver couldn’t locate the element”); | |

|  |  |
| --- | --- |
| 11 | } |

**#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.*

|  |  |  |
| --- | --- | --- |
| 1 | for (String handle : driver.getWindowHandles()) { | |
| 2 | try { |

|  |  |
| --- | --- |
| 3 | driver.switchTo().window(handle); |
| 4 | } catch (NoSuchWindowException e) { | |

|  |  |  |
| --- | --- | --- |
| 5 | System.out.println(“An exceptional case”); | |
| 6 | } |

|  |  |
| --- | --- |
| 7 | } |

**#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:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | driver.switchTo().frame("frame\_11"); | |

|  |  |
| --- | --- |
| 3 | } 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.*

|  |  |
| --- | --- |
| 1 | try { |
| 2 | WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10)); | |

|  |  |  |
| --- | --- | --- |
| 3 | wait.Until(ExpectedConditions.frameToBeAvaliableAndSwitchToIt(frame\_11)); | |
| 4 | try { |

|  |  |  |
| --- | --- | --- |
| 5 | driver.switchTo().frame("frame\_11"); | |
| 6 | } catch (WebDriverException e) { |

|  |  |  |
| --- | --- | --- |
| 7 | System.out.println(“An exceptional case”); | |
| 8 | } |

|  |  |  |
| --- | --- | --- |
| 9 | } catch (TimeOutException e) { | |
| 10 | | System.out.println(“WebDriver couldn’t locate the frame”); | |

|  |  |
| --- | --- |
| 11 | } |

**#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:

|  |  |
| --- | --- |
| 1 | try { |
| 2 | driver.switchTo().alert().accept(); | |

|  |  |
| --- | --- |
| 3 | } 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.*

|  |  |
| --- | --- |
| 1 | try { |
| 2 | WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10)); | |

|  |  |  |
| --- | --- | --- |
| 3 | wait.Until(ExpectedConditions.alertIsPresent()); | |
| 4 | try { |

|  |  |
| --- | --- |
| 5 | driver.switchTo().alert().accept(); |
| 6 | } catch (NoAlertPresentException e) { | |

|  |  |  |
| --- | --- | --- |
| 7 | System.out.println(“An exceptional case”); | |
| 8 | } |

|  |  |  |
| --- | --- | --- |
| 9 | } catch (TimeOutException e) | |
| 10 | | System.out.println(“WebDriver couldn’t locate the Alert”); | |

|  |  |
| --- | --- |
| 11 | } |

**#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***

|  |  |
| --- | --- |
| 1 | try { |
| 2 | clickXPathButtonAndWait("//button[@type='button']"); | |

|  |  |  |
| --- | --- | --- |
| 3 | } catch (InvalidSelectorException e) { | |
| 4 | } |

**#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.

|  |  |  |
| --- | --- | --- |
| 1 | driver.findElement(By.id("submit")).click(); | |
| 2 | Exception Handling: |

|  |  |
| --- | --- |
| 3 | try { |
| 4 | driver.findElement(By.id("submit")).click(); | |

|  |  |
| --- | --- |
| 5 | } 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.*

|  |  |
| --- | --- |
| 1 | try { |
| 2 | WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10)); | |

|  |  |  |
| --- | --- | --- |
| 3 | wait.Until(ExpectedConditions.visibilityOfElementLocated(By.id(”submit”)); | |
| 4 | try { |

|  |  |  |
| --- | --- | --- |
| 5 | driver.findElement(By.id("submit")).click(); | |
| 6 | } catch (WebDriverException e) { |

|  |  |  |
| --- | --- | --- |
| 7 | System.out.println(“Exceptional case”); | |
| 8 | } |

|  |  |  |
| --- | --- | --- |
| 9 | } catch (TimeOutException e) | |
| 10 | | System.out.println(“WebDriver couldn’t find this element visible”); | |

|  |  |
| --- | --- |
| 11 | } |

**#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.*

|  |  |
| --- | --- |
| 1 | try { |
| 2 | WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(10)); | |

|  |  |  |
| --- | --- | --- |
| 3 | wait.Until(ExpectedConditions. elementToBeClickable(By.id(”swift”)); | |
| 4 | try { |

|  |  |  |
| --- | --- | --- |
| 5 | Select dropdown = new Select(driver.findElement(By.id("swift"))); | |
| 6 | } catch (WebDriverException e) { |

|  |  |  |
| --- | --- | --- |
| 7 | System.out.println(“Exceptional case”); | |
| 8 | } |

|  |  |  |
| --- | --- | --- |
| 9 | } catch (TimeOutException e) | |
| 10 | | System.out.println(“WebDriver found that this element was not selectable.”); | |

|  |  |
| --- | --- |
| 11 | } |

**#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.*

|  |  |  |
| --- | --- | --- |
| 1 | WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(30)); | |
| 2 |  |

|  |  |  |
| --- | --- | --- |
| 3 | wait.until(webDriver -> ((JavascriptExecutor)webDriver).executeScript("return document.readyState").equals("complete")); | |
| 4 |  |

|  |  |
| --- | --- |
| 5 | 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*

|  |  |
| --- | --- |
| 1 | @BeforeSuite |
| 2 | public void setUp() throws MalformedURLException { | |

|  |  |  |
| --- | --- | --- |
| 3 | WebDriver driver = new FirefoxDriver(); | |
| 4 | } |

|  |  |
| --- | --- |
| 5 | @AfterSuite |
| 6 | public void testDown() { | |

|  |  |  |
| --- | --- | --- |
| 7 | driver.quit(); | |
| 8 | } |

**#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,*

|  |  |  |
| --- | --- | --- |
| 1 | | try { |
| 2 | driver.findElement(By.xpath(“//\*[contains(@id,firstname’)]”)).sendKeys(“Aaron”); | | |

|  |  |
| --- | --- |
| 3 | } 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‘.

[**Next Tutorial #20**:](https://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/) In the upcoming tutorial, we would discuss the **various types of testing frameworks available**. We would also study the pros and cons of using a fledged framework approach in automation testing. We would discuss in detail about the Test data-driven framework.

***Please post your queries, related to handling exception in Selenium WebDriver, if you have any.***

**Recommended Reading**

* [Java Exceptions and Procedures to Handle Exception](https://www.softwaretestinghelp.com/java/java-exceptions/)
* [How to Handle Alerts/Popups in Selenium WebDriver - Selenium Tutorial #16](https://www.softwaretestinghelp.com/handle-alerts-popups-selenium-webdriver-selenium-tutorial-16/)
* [How to Handle Windows Pop-up Using AutoIt and Selenium Automation Testing](https://www.softwaretestinghelp.com/autoit-tutorial-to-download-write-autoit-script/)
* [Cucumber Selenium Tutorial: Cucumber Java Selenium WebDriver Integration](https://www.softwaretestinghelp.com/selenium-webdriver-cucumber-selenium-tutorial-31/)
* [Integration of Selenium with JMeter](https://www.softwaretestinghelp.com/integrating-jmeter-selenium/)
* [Introduction to Selenium WebDriver – Selenium Tutorial #8](https://www.softwaretestinghelp.com/selenium-webdriver-selenium-tutorial-8/)
* [30+ Best Selenium Tutorials: Learn Selenium With Real Examples](https://www.softwaretestinghelp.com/selenium-tutorial-1/)
* [Part-time Freelancing Job Opportunity for Selenium Experts](https://www.softwaretestinghelp.com/part-time-freelancing-job-opportunity-for-selenium-experts/)

Categories [Selenium Tutorials](https://www.softwaretestinghelp.com/category/selenium-tutorials/) Post navigation

[Handling Web Tables, Frames, and Dynamic Elements in Selenium Script – Selenium Tutorial #18](https://www.softwaretestinghelp.com/selenium-tutorial-18/)

[Most Popular Test Automation Frameworks with Pros and Cons of Each – Selenium Tutorial #20](https://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/)

**26 thoughts on “Top 10 Selenium Exceptions and How To Handle These (Exact Code)”**

1. *Ahmed*