Selenium Exceptions

1) NoSuchElementException : FindBy method can't find the element.

2) StaleElementReferenceException : This tells that element is no longer appearing on the DOM page.

3) TimeoutException: This tells that the execution is failed because the command did not complete in enough time.

4) 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

5) ElementNotSelectableException: Thrown to indicate that may be the element is disabled, and so is not able to select.

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

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

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

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

10) 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.

1) NoSuchElementException :

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)
```

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) StaleElementReferenceException:

1.The element has been deleted entirely.

2.The element is no longer attached to the DOM.

3.You click something that loads a new page asynchronously or at least changes it.

4.You immediatelly (before the page load could finish) search for an element ... and you find it!

5.The page finally unloads and the new one loads up.

6.You try to access your previously found element, but now it's stale, even though the new page contains it, too.

How to fix or handle it

Solution 1–
The Most way to handle this is to refresh the page, On refreshing it, most of the time driver found the element, But it's not the perfect solution-

```
Driver.navigate().refersh();
Driver.findElement(By.id("property")).click();
Driver.navigate().refersh();
```

```
Driver.findElement(By.id("property")).click();
```

Solution 2-
The second option you can try is to place your driver interaction command inside some loop and to repeat that for a couple of times, 2-3 times, refer below code.

however this method is also but hardcoded and not generic, it can also fail, if even after 3 attempt element is not found however its best to use the third option which is placing Do-While loop

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

{

try{
```

```
driver.findElement("Property").click();

break;
}
}

catch(Exception e)

{
```

```
public void StaleElementHandleByID (String elementID)
{
   int count = 0;
   boolean clicked = false;
   while (count < 4 && !clicked)
   {
      try
      {
         WebElement yourSlipperyElement= driver.findElement(By.id(elementID));
         yourSlipperyElement.click();
         clicked = true;
      }
      catch (StaleElementReferenceException e)
      {
         e.toString();
         System.out.println("Trying to recover from a stale element :" + e.getMessage());
         count = count+1;
      }
   }
}
```

Solution 3–

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)
```

Solution 4–Wait

```
Select selectLocation = new Select(new WebDriverWait(driver, 30)
   .until(ExpectedConditions
   .presenceOfElementLocated(By.id("filterForm:selectLocation"))));
```

3) TimeoutException:

Solution 1

```
@Test
public void testEmptyCaption() throws Exception {
   openTestURL();
   // Wait for the loading bar to disappear before taking the screenshot.
   try {
      waitUntil(ExpectedConditions.invisibilityOfElementLocated(By.className("v-loading-
indicator")), 5);
   } catch (TimeoutException e) {
   }
   compareScreen("table-empty-caption");
}
```

Solution 2

```
WebDriver driver = new FirefoxDriver();
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
driver.get("http://somedomain/url_that_delays_loading");
WebElement myDynamicElement = driver.findElement(By.id("myDynamicElement"));
```

Solution 3

By doing some heavy research, I was able to find the solution by using:

```
wait.until(ExpectedConditions.presenceOfAllElementsLocatedBy(By.finder));
```

The key was to find all the elements present in the part where i was going to validate if it was available. This solved the Time out's error:

As for the Element is no longer attached to the DOM, and the validations, instead of using the .IsDisplayed or .IsEnabled which DON'T returns "false" but an exception, i was able to validate this using

```
findElements(By.finder).size()>0
```

This returns false or true if the element size is displaying or not.

4) ElementNotVisibleException

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");
}
```

5) ElementNotSelectableException

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.”);
```

6) NoSuchWindowException

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”);
}
}
```

7) NoSuchFrameException:

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 (TimeOutException e) {
System.out.println("WebDriver couldn't locate the frame");
}
```

8) NoAlertPresentException:

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");
}
```

9) InvalidSelectorException:

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 o
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) {
}
```

10) NoSuchSessionException:

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();
}
```

Stale Element Reference Exception:

1.https://stackoverflow.com/questions/17174515/how-to-resolve-stale-element-exception-if-element-is-no-longer-attached-to-the

2.https://automatorsworld.com/2018/10/19/handle-stale-element-reference-exception-selenium-webdriver/

3.https://stackoverflow.com/questions/4846454/selenium-webdriver-staleelementreferenceexception

4.https://gist.github.com/djangofan/5112655

Dynamic xpath:

https://sqa.stackexchange.com/questions/18342/how-to-handle-dynamic-changing-ids-in-xpath

Selenium Exceptions:

https://www.softwaretestinghelp.com/exception-handling-framework-selenium-tutorial-19/

https://www.guru99.com/exception-handling-selenium.html

https://www.toolsqa.com/selenium-webdriver/exception-handling-selenium-webdriver/