**1. What are the different types of Exceptions in Selenium?**

**Answer:**During automation in Selenium WebDriver, we come across various exceptions & we need to deal with them. Below is the list of various exceptions occur in selenium webdriver.

There are main three types of Exceptions in Selenium WebDriver –

**Checked Exceptions** – These Exceptions can be handled during compile time. If they are not handled, it gives compile time error. Example- FileNotFoundException, IOException etc.

**Unchecked Exceptions** – These exceptions can not be handled during compile time & they got caught at run time. Example – ArrayIndexOutOfBoundException.

**Error** – Errors which can not be handled by using even try catch block. Example -Assertion Error.

**Common Exceptions in Selenium**

1. ElementNotVisibleException - This type of Selenium exception occurs when an existing element in DOM has a feature set as hidden.

2. ElementNotSelectableException - This Selenium exception occurs when an element is presented in the DOM, but you can be able to select. Therefore, it is not possible to interact.

3. NoSuchElementException - This Exception occurs if an element could not be found in DOM.

4. NoSuchFrameException - This Exception occurs if the frame target to be switched to does not exist.

5. NoAlertPresentException - This Exception occurs when you switch to no presented alert.

6. NoSuchWindowException - This Exception occurs if the window target to be switch does not exist.

7. StaleElementReferenceException  - This Selenium exception occurs when the web element is detached from the current DOM.

8. SessionNotFoundException - The WebDriver is acting after you quit the browser.

9. TimeoutException - Thrown when there is not enough time for a command to be completed. For Example, the element searched wasn't found in the specified time.

10. WebDriverException - This Exception takes place when the WebDriver is acting right after you close the browser.

11. ConnectionClosedException - This type of Exception takes place when there is a disconnection in the driver.

12. ElementNotInteractableException - This Selenium exception is thrown when any element is presented in the DOM. However, it is impossible to interact with such an element.

13. InvalidArgumentException - It occurs when an argument does not belong to the expected type.

14. SessionNotCreatedException - It happens when a new session could not be successfully created.

15. ElementNotVisibleException - If selenium tries to find an element but the element is not visible within the page

16. NoSuchAttributeException - While trying to get attribute value but the attribute is not available in DOM.

Read About Java Exceptions here: <https://www.automationtestinginsider.com/2019/08/exception-handling-in-java.html>

**2. How to handle Selenium WebDriver Exceptions?**

**Answer:**  Handling Exceptions In Selenium WebDriver

Following are a few standard ways using which one can handle Exceptions in Selenium WebDriver:

**Try-catch:** This method can catch Exceptions by using a combination of the try and catch keywords. Try indicates the start of the block, and Catch is placed at the end of the try block to handle or resolve the Exception. The code that is written within the Try/Catch block is referred to as “protected code.” The following code represents the syntax of Try/Catch block –

try

   {

     // Some code

   }

catch(Exception e)

  {

     // Code for Handling the exception

  }

**Multiple catch blocks**: There are various types of Exceptions, and one can expect more than one exception from a single block of code. Multiple catch blocks are used to handle every kind of Exception separately with a separate block of code. One can use more than two catch blocks, and there is no limitation on the number of catch blocks. The code below represents the syntax of multiple catch blocks –

try

   {

      //Some code

   }

catch(ExceptionType1 e1)

   {

     //Code for Handling the Exception 1

   }

catch(ExceptionType2 e2)

   {

     //Code for Handling the Exception 2

   }

**Throw/Throws:** When a programmer wants to generate an Exception explicitly, the Throw keyword is used to throw Exception to runtime to handle it. When a programmer is throwing an Exception without handling it, then he/she needs to use Throws keyword in the method signature to enable the caller program to understand the exceptions that might be thrown by the method. The syntax for Throws is as follows:

public static void anyFunction() throws Exception

{

try

   {

     // write your code here

   }

catch (Exception e)

   {

      // Do whatever you wish to do here

      // Now throw the exception back to the system

      throw(e);

   }

}

**Multiple Exceptions:** One can mention various Exceptions in the throws clause. Refer to the example below:

public static void anyFunction() throws ExceptionType1, ExceptionType2

{

 try

   {

     // write your code here

   }

 catch (ExceptionType1 e1)

   {

    // Code to handle exception 1

   }

 catch (ExceptionType1 e2)

   {

    // Code to handle exception 2

   }

}

**Finally:** The Final keyword is used to create a block of code under the try block. This final code block always executes irrespective of the occurrence of an exception

try

  {

    //Protected code

  }

catch(ExceptionType1 e1)

  {

    //Catch block

  }

catch(ExceptionType2 e2)

  {

    //Catch block

  }

catch(ExceptionType3 e3)

  {

    //Catch block

  }

finally

   {

     //The finally block always executes.

   }

One can also use the following methods to display Exception Information:

printStackTrace(): It prints the stack trace, name of the exception, and other useful description

toString(): It returns a text message describing the exception name and description

getMessage(): It displays the description of the exception

**3. What are the different types of exceptions you have faced in Selenium WebDriver?**

**Answer:**  Please refer question#1

**4. What are the types of exceptions which will appear while finding elements?**

**Answer:**

1. ElementNotVisibleException - This type of Selenium exception occurs when an existing element in DOM has a feature set as hidden.

2. ElementNotSelectableException - This Selenium exception occurs when an element is presented in the DOM, but you can be able to select. Therefore, it is not possible to interact.

3. NoSuchElementException - This Exception occurs if an element could not be found in DOM.

4. StaleElementReferenceException  - This Selenium exception occurs when the web element is detached from the current DOM.

5. ElementNotInteractableException - This Selenium exception is thrown when any element is presented in the DOM. However, it is impossible to interact with such an element.

6. ElementNotVisibleException - If selenium tries to find an element but the element is not visible within the page

7. NoSuchAttributeException - While trying to get attribute value but the attribute is not available in DOM.

**5. What is a null pointer exception ? Is it checked or not?**

**Answer:**NullPointerException is a unchecked exception(Run time).NullPointerException (usually) occurs because there is some bug in your code. If you expect a NullPointerException to be thrown, the correct solution is to fix the bug rather than to handle the exception.

NullPointerException doesn't force us to use catch block to handle it.

**6. What is StaleElementException? When does it occur? How do you handle it?**

**Answer:**Stale means old, decayed, no longer fresh. Stale Element means an old element or no longer available element. Assume there is an element that is found on a web page referenced as a WebElement in WebDriver. If the DOM changes then the WebElement goes stale. If we try to interact with an element which is staled then the StaleElementReferenceException is thrown.

The two reasons for Stale element reference are

The element has been deleted entirely.

The element is no longer attached to the DOM.

**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.name(“name”)).click();

Driver.navigate().refersh();

Driver.findElement(By.name(“name”)).click();

**Solution 2:**

If an element is not attached to DOM then you could try using ‘try-catch block’ within ‘for loop’

// Using for loop, it tries for 3 times.

// If the element is located for the first time then it breaks from the for loop nad comeout of the loop

for(int i=0; i<=2;i++){

  try{

     driver.findElement(By.xpath("xpath")).click();

     break;

  }

  catch(Exception e){

     Sysout(e.getMessage());

  }

}

**Solution 3:**

Wait for the element till it gets available

wait.until(ExpectedConditions.presenceOfElementLocated(By.id("id")));

**Solution 4:**

We can handle Stale Element Reference Exception by using POM.

We could avoid StaleElementException using POM. In POM, we use initElements() method which loads the element but it won’t initialize elements. initElements() takes latest address. It initializes during run time when we try to perform any action on an element. This process is also known as Lazy Initialization.

**7. NoSuchElementException and ElementNotVisibleException  what is difference?**

**Answer:**

1. NoSuchElementException - This Exception occurs if an element could not be found in DOM.

2. ElementNotVisibleException - This type of Selenium exception occurs when an existing element in DOM has a feature set as hidden.

**8. Why we are getting class not found exception?**

**Answer:** The ClassNotFoundException is thrown when the Java Virtual Machine (JVM) tries to load a particular class and the specified class cannot be found in the classpath. The ClassNotFoundException is a checked exception and thus, must be declared in a method or constructor’s throws clause.

The following example tries to load a class using the forName method. However, the specified class name cannot be found and thus, a ClassNotFoundException is thrown.

**public** **class** **ClassNotFoundExceptionExample** {

**private** **static** **final** String CLASS\_TO\_LOAD = "main.java.Utils";

**public** **static** **void** **main**(String[] args) {

**try** {

Class loadedClass = Class.forName(CLASS\_TO\_LOAD);

System.out.println("Class " + loadedClass + " found successfully!");

}

**catch** (ClassNotFoundException ex) {

System.err.println("A ClassNotFoundException was caught: " + ex.getMessage());

ex.printStackTrace();

}

}

}

**How to deal with the ClassNotFoundException**

Verify that the name of the requested class is correct and that the appropriate .jar file exists in your classpath. If not, you must explicitly add it to your application’s classpath.

In case the specified .jar file exists in your classpath then, your application’s classpath is getting overriden and you must find the exact classpath used by your application.

In case the exception is caused by a third party class, you must identify the class that throws the exception and then, add the missing .jar files in your classpath.

**9. Diff between classnotfound exception and nodeffoundexception?**

**Answer:**  ClassNotFoundException is an exception that occurs when you try to load a class at run time using Class. forName() or loadClass() methods and mentioned classes are not found in the classpath. NoClassDefFoundError is an error that occurs when a particular class is present at compile time, but was missing at run time.

ClassNotFoundException is raised in below program as class “ATI” is not found in classpath.

// ClassNotFoundException

**public** **class** **Example** {

**public** **static** **void** **main**(String args[]) {

**try**

{

Class.forName("ATI");

}

**catch** (ClassNotFoundException ex)

{

ex.printStackTrace();

}

}

}

**Output**

java.lang.ClassNotFoundException: ATI

Below program will be successfully compiled and generate two classes Example1. class and Test.class

Now remove Example1.class file and run Test.class.

At Java runtime NoClassDefFoundError will be thrown.

// NoClassDefFoundError

**class** **Example1**

{

**void** **greeting**()

{

System.out.println("hello!");

}

}

**class** **Test** {

**public** **static** **void** **main**(String args[])

{

Example1 obj = **new** Example1();

obj.greeting();

}

}

**Output**

java.lang.NoClassDefFoundError : Example1