Exceptions in Selenium | 40 Exceptions

Table of content

Exceptions in java

An Exception is an unplanned surprise that occurs during execution of program, the exception has power to stop the program execution.

Without a doubt, the exception stops the execution, but we are more powerful than the exception, so we can handle the exception without stopping the execution of the program.

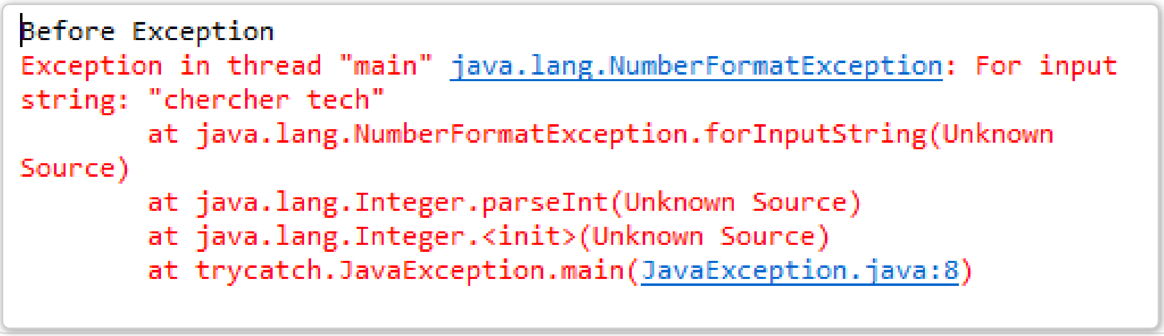
We can obtain the normal flow of the program by handling the exception, and we can provide a more meaningful message without stopping the execution of the program.

If these exceptions are not handled properly, the remaining program will not be executed.

```
public class JavaException {
   public static void main(String[] args)
   {
      System.out.println("Before Exception");
      // below code throws Exception
      Integer intValue = new Integer("chercher tech");
      System.out.println("Converted value : "+intValue);
      System.out.println("After Exception");
   }
}
```

In the above program, the line *Integer("chercher tech")* tries to convert the alphabet string into a number. Converting the alphabets into a number is not possible, so the program throws a java.lang.NumberFormatException.

The code present after that particular line will not be executed as exception stops the total program execution.



```
Before Exception
Exception in thread "main" java.lang.NumberFormatException: For input
string: "chercher tech"
        at java.lang.NumberFormatException.forInputString(Unknown
Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.<init>(Unknown Source)
        at trycatch.JavaException.main(JavaException.java:8)
```

Exception Architecture in Java

Throwable is super most class for Exceptions in Java; Throwable can be categorized as Error and Exception.

Error : Error is a serious condition; the application must not try to handle the error. Errors are mainly caused by the environment in which the application is running. Errors are not checked at compile-time and do not have to be (but can be) caught or handled.

The coder should not worry about errors at all in java, as errors indicate the problems with the system or environment. Below are a few Errors :

OutOfMemoryError

StackOverFlowError

LinkageError

Exception : Exceptions are caused by the application, Exceptions are derived from Throwable class. Exceptions can be categorized into two:

Checked Exceptions also are known as Compile time Exception

Un-Checked Exceptions also known as Runtime exceptions

Checked Exceptions, aka Compile Time Exceptions : Checked Exceptions are the exceptions that are verified at compile time, compiles itself senses that particular code can raise exceptions.

If the code within a method throws a checked exception, then the method must either handle the exception, or it must delegate the exception using throws keyword if we do not do either of them, then the compiler will not allow you to compile the program.

Below are few checked exceptions :

IOException

EOFException

Exception

MalFormedURLException

IntruptedException

Unchecked Exceptions, aka Runtime Exceptions : The classes which extend RuntimeException are known as unchecked exceptions. Unchecked exceptions occur during the run time of the program. The compiler will not be able to detect the unchecked exception during compile time.

Below are few unchecked exceptions :

ArithmaticException

NullPointerException

IndexOutOfBoundsException

ClassCastException

ArrayIndexOutOfBoundsException

NoSuchElementException

NoSuchFrameException

NumberFormatException

Difference between Error and Exception

Exception and Error both are subclasses of Throwable class and both halts the program execution

> User should never design an application to throw Error, but the user can design an application which throws Exception

Errors in java are of type java.lang.Error, whereas Exceptions in java are of type java.lang.Exception

All errors in java are unchecked type; Exceptions include both checked as well as unchecked type.

Errors happen at run time; the compiler will not have knowledge of it. Checked exceptions are known to the compiler, whereas unchecked exceptions are not known to compiler because they occur at runtime.

It is impossible to handle errors, but we can handle exceptions through try..catch..finally blocks.

Errors are mostly caused by the environment in which the application is running. Exceptions are mainly caused by the application itself.

StackOverflowError, OutOfMemoryError are few examples for Error, IOException, NullpointerException is few examples for Exception.

Methods Present in Exceptions

Methods present in the Exceptions helps the user to get different details of the Exception. Below are a few useful methods present in Exceptions.

1. getMessage() / toString() : Returns the detailed message string, most of the time a reason why this exception occurred

2. getLocalizedMessage() : returns the localized message according the language. This is the local version of getMessage(). Creates a localized description of this Exception.

Subclasses may override this method in order to produce a locale-specific message. For subclasses that do not override this method, the default implementation returns the same result as getMessage().

3. getCause() : Returns the cause of the exception or null if the cause is nonexistent or unknown.

4. getStackTrace() : This method fetches the detailed exception details like which line caused the exception, which all are methods got affected, what is the error message, and a few more details.

5. printStackTrace() : This method is similar to getStackTrace(); the only difference is this method prints the details rather than fetching the details.

Try..Catch..Finally

You might have come across situations where your program got stopped abruptly throwing some exception. Exception (recap): Exception is abnormal behavior of the code, or it happens because something is wrong with our code or with our system.

Here, abnormal behavior means something or wrong with our code, something like we are trying to find some element, but the element is not present in the webpage; the program throws an exception when we try to access a file that is not present in our local system, etc..

In this tutorial, you are going to learn try and catch block based only on Selenium WebDriver.

try : try block is nothing but a block which contains a specific code and that certain code may or may not behave abnormally when the code behaves abnormally the program throws an exception, but when it behaves normally, it will not throw any exception.

In layman terms, try block is nothing but a room in the hospital which is little sensitive, and we have patient inside the particular room, when something unexpected happens in that room the hospital will be ready to handle the situation.

When an exception occurs in the try block, the code after the exception in the try block will not be executed.

```
try{
   // code that we need to monitor
   }
```

catch : The catch block is nothing but a block of code, so whenever something abnormal happens in the try block the associated catch block will be executed, the catch block will have code which will efficiently handle that particular exceptional situation

In the catch block, the user should mention what is the exception that we should handle because we cannot handle all the exceptions as different exceptions will require different code to handle the situation.

The catch block will not be executed if there is no exception occurs in the try block

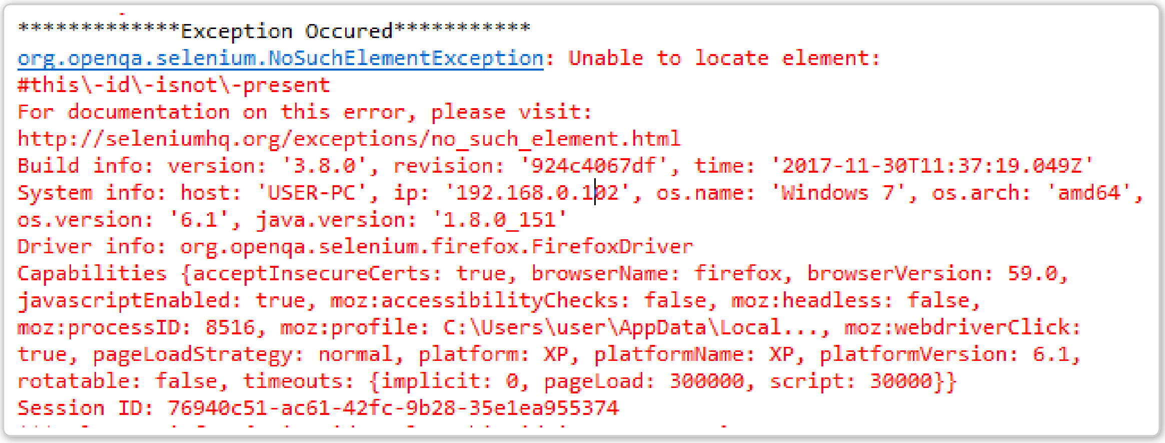
```
try{
   // code that we need to monitor
   }
   catch(Exception e)
   {
      // what should we do if something abnormal occurs
   }
```

Program for simple try and catch block; there is no element present in the page with id='this-id-is-not-present'

```
public static void main(String[] args) {
      // set the geckodriver.exe property
      System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
      // open firefox
      WebDriver driver = new FirefoxDriver();
      driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
      driver.get("https://chercher.tech/java/index-selenium-webdriver");

      try{
         // no element is present with such id
         driver.findElement(By.id("this-id-is-not-present")).click();

      }catch(Exception e){
         System.out.println("*************Exception Occurred***********");
         e.printStackTrace();
      }
   }
```



```
*************Exception Occured***********
org.openqa.selenium.NoSuchElementException: Unable to locate element:
#this\-id\-isnot\-present
For documentation on this error, please visit:
http://seleniumhq.org/exceptions/no_such_element.html
Build info: version: '3.8.0', revision: '924c4067df', time: '2017-11-30T11:37:19.049Z'
System info: host: 'USER-PC', ip: '192.168.0.102', os.name: 'Windows 7', os.arch: 'amd64',
os.version: '6.1', java.version: '1.8.0_151'
Driver info: org.openqa.selenium.firefox.FirefoxDriver
Capabilities {acceptInsecureCerts: true, browserName: firefox, browserVersion: 59.0,
javascriptEnabled: true, moz:accessibilityChecks: false, moz:headless: false,
moz:processID: 8516, moz:profile: C:\Users\user\AppData\Local..., moz:webdriverClick:
true, pageLoadStrategy: normal, platform: XP, platformName: XP, platformVersion: 6.1,
rotatable: false, timeouts: {implicit: 0, pageLoad: 300000, script: 30000}}
Session ID: 76940c51-ac61-42fc-9b28-35e1ea955374
```

The above program throws an exception, and the catch will be executed, the content in the catch block will be executed.

finally : finally block is also a normal block which contains code but finally block will be executed irrespective of whether an exception finally : finally block is also a normal block which contains code but finally block will be executed irrespective of whether an exception occurred or not in the try block

Basic rules of finally :

finally block will be used to perform code cleanup activities like : disconnecting browser, closing all files, closing the browser.

try block is must to write finally block

Only one finally block is allowed with the try block

When finally block will not execute :

When System.exit() code occurs in try or in a catch block

When JVM crashes

When try block is executed infinitely without breaking

When the user stopped the execution (just for fun).

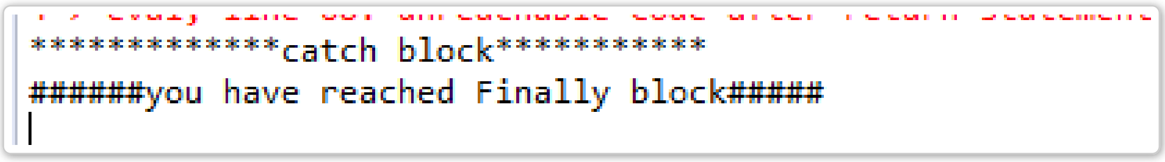
In the below program, the code present in try block will throw an exception, and the catch block will be executed, then finally block gets executed.

Program with try..catch..finally.

```java
public static void main(String[] args) {
   // set the geckodriver.exe property
   System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
   // open firefox
   WebDriver driver = new FirefoxDriver();
   driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
   driver.get("https://chercher.tech/java/index-selenium-webdriver");

   try{
      // no element is present with such id
      driver.findElement(By.id("this-id-isnot-present")).click();

   }catch(Exception e){
      System.out.println("*************catch block***********");
   }finally{
      System.out.println("######you have reached Finally block#####");
   }
}
```

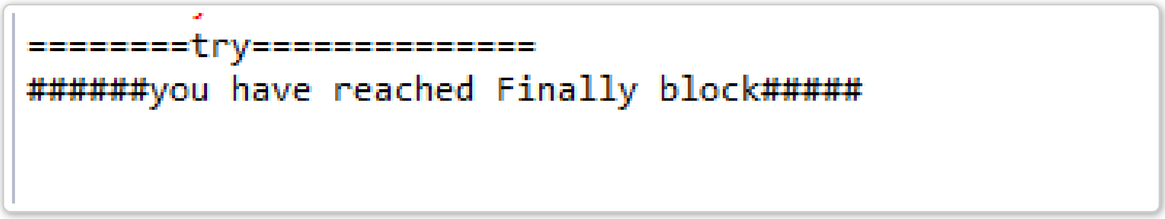


```
*************catch block***********
######you have reached Finally block#####
```

In the below program, the code present in try block will not throw any exception, so the catch block will not be executed as the catch block gets executed when there is an exception in the try block, at last finally block will be executed after the try block.

```
public static void main(String[] args) {
   // set the geckodriver.exe property
   System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
   // open firefox
   WebDriver driver = new FirefoxDriver();
   driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
   driver.get("https://chercher.tech/java/index-selenium-webdriver");
   try{
      System.out.println("========try==============");

   }catch(Exception e){
      System.out.println("*************catch block***********");
   }finally{
      System.out.println("######you have reached Finally block#####");
   }
}
```



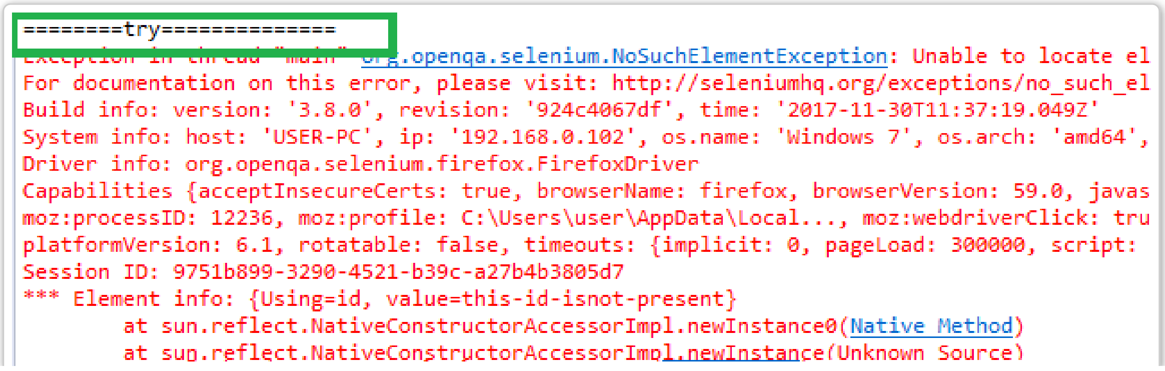
```
========try===============
######you have reached Finally block#####
```

finally block without catch block : Try block expects either catch block or finally block to present or both catch and finally blocks to present but writing both catch and finally block is not mandatory.

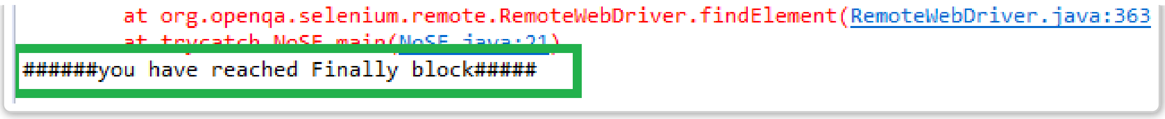
Program for try block without a catch block

```
public static void main(String[] args) {
   // set the geckodriver.exe property
   System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
   // open firefox
   WebDriver driver = new FirefoxDriver();
   driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
   driver.get("https://chercher.tech/java/index-selenium-webdriver");
   try{
      System.out.println("========try==============");
      // no element is present with such id
      driver.findElement(By.id("this-id-isnot-present")).click();

   }finally{
      System.out.println("######you have reached Finally block#####");
   }
}
```



```
========try===============
Exception in thread "main" org.openqa.selenium.NoSuchElementException: Unable to locate el
For documentation on this error, please visit: http://seleniumhq.org/exceptions/no_such_el
Build info: version: '3.8.0', revision: '924c4067df', time: '2017-11-30T11:37:19.049Z'
System info: host: 'USER-PC', ip: '192.168.0.102', os.name: 'Windows 7', os.arch: 'amd64',
Driver info: org.openqa.selenium.firefox.FirefoxDriver
Capabilities {acceptInsecureCerts: true, browserName: firefox, browserVersion: 59.0, javas
moz:processID: 12236, moz:profile: C:\Users\user\AppData\Local..., moz:webdriverClick: tru
platformVersion: 6.1, rotatable: false, timeouts: {implicit: 0, pageLoad: 300000, script:
Session ID: 9751b899-3290-4521-b39c-a27b4b3805d7
*** Element info: {Using=id, value=this-id-isnot-present}
        at sun.reflect.NativeConstructorAccessorImpl.newInstance0(Native Method)
        at sun.reflect.NativeConstructorAccessorImpl.newInstance(Unknown Source)
```



```
        at org.openqa.selenium.remote.RemoteWebDriver.findElement(RemoteWebDriver.java:363
        at trycatch.NoSE.main(NoSE.java:21)
######you have reached Finally block#####
```

In the above image, you can see that finally and block executed, and there is no catch block in the program.

Basic rules of try..catch..finally :

The code present in try block may or may not raise the exception

try block or catch or finally blocks cannot exits alone

catch or finally, or both should follow try block

the catch block is optional when finally block is present

finally block is optional when the catch block is present

There is no block which executes when only an exception is not occurred in java (but exists in python language)

We can't have catch or finally block without a try statement.

We can't write any code between try..catch..finally block.

try..catch blocks can be nested (try inside another try or catch or finally)

We can have only one finally block with a try block.

Purpose of try..catch..finally

The purpose of the try-catch block is to continue the execution of the program without stopping the execution whenever there is an exception Scenario :

1. Open the browser and navigate to **https://chercher.tech/java/index-selenium-webdriver**
2. Click the element whose id value is ' this-id-isnot-present'
3. Close the browser.

Program without try..catch..finally

```
public static void main(String[] args) {
   // set the geckodriver.exe property
   System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
   // open firefox
   WebDriver driver = new FirefoxDriver();
   driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
```

```
    driver.get("https://chercher.tech/java/index-selenium-webdriver");
    // no element is present with such id
    driver.findElement(By.id("this-id-isnot-present")).click();
    driver.close();
}
```

The above program did not close the browser, i.e., as there no element with id='this-id-isnot-present', so **find element** throws an exception, and the program stops when there is an exception. The control of the program never reaches the `driver.close()` command

Program with try..catch..finally

```
public static void main(String[] args) {
    // set the geckodriver.exe property
    System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
    // open firefox
    WebDriver driver = new FirefoxDriver();
    driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
    driver.get("https://chercher.tech/java/index-selenium-webdriver");

    try{
        // no element is present with such id
        driver.findElement(By.id("this-id-isnot-present")).click();

    }catch(Exception e){
        // we can write the code when there is no element with given locator
        System.out.println("No element present using 'id=this-id-isnot-present'");
    }
    finally{
        driver.close();
    }
}
```

The above code closes the browser after checking whether the element is present or not.

Think about why did I place the driver.close() in finally block ?

if you know an answer enter your answer in the comment section of the page

Implementing Class Constructor locator in Selenium

Exact Exception in the catch block

We always should try(normal English) to write the exact exception that we are looking for in the try block.

For example, in the below code, I am trying to find an element, **click** the element in selenium webdriver, and I am doubtful that the element may or may not exist if the element is not present I want to print that "element is not there".

We know that selenium webdriver throws **NoSuchElementException** if there is no element present with a given locator.

```
try{
   // no element is present with such id
   driver.findElement(By.id("this-id-isnot-present")).click();

}catch(Exception e){
   System.out.println("element is not there");
}
```

In above code, Exception e the catch block will accept all the exceptions. For instance if the element is there on the web page but the element is disabled, then selenium webdriver will throw <href='exceptions-selenium-webdriver#invalidelementstateexception' target="_blank" title=" invalid elemnt state exception">InvalidElementStateException </href='exceptions-selenium-webdriver#invalidelementstateexception'>but our catch block will catch this exception as well, because Exception is parent class of exceptions.

If you remember, our aim was to check whether the element is there on the webpage or not, but we get the conclusion that the element is not present even though the element is present.

This is the reason why we should write a specific exception in the catch block to server our purpose.

Corrected code

```
try{
   // no element is present with such id
   driver.findElement(By.id("this-id-isnot-present")).click();

}catch(NoSuchElementException e){
   System.out.println("element is not there");
}
```

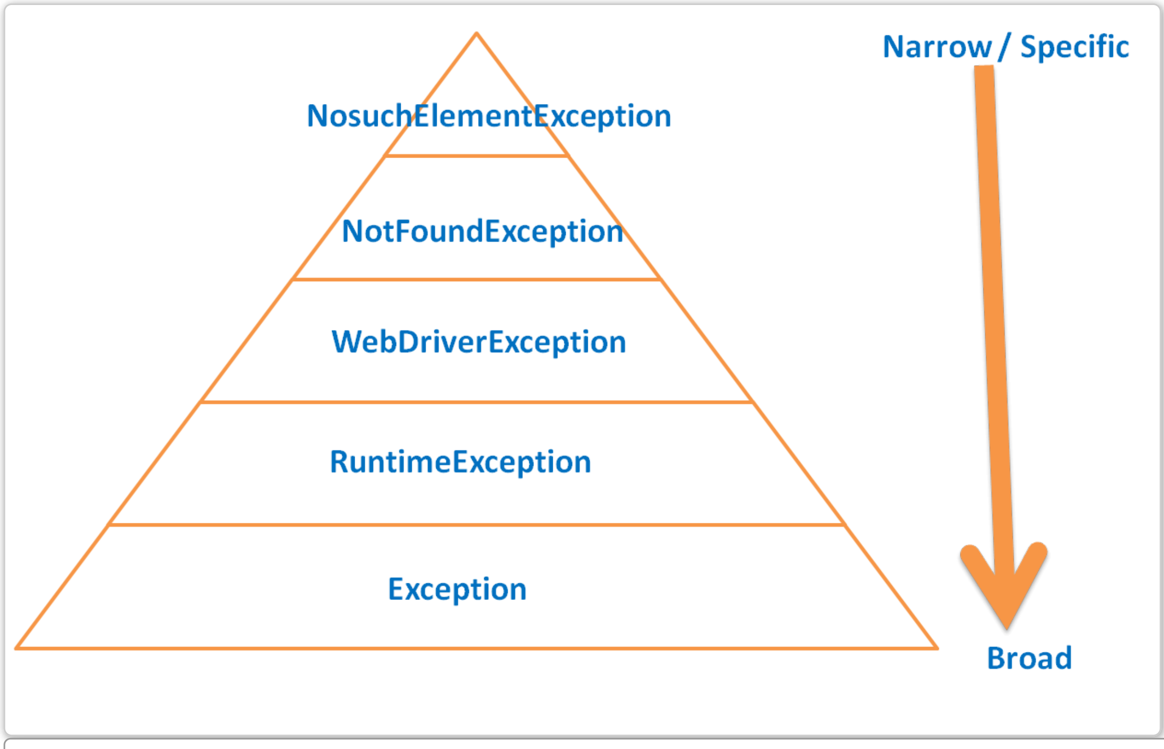
Multiple catch blocks with a try block

Java lets the user to write multiple catch block, multiple catch block is nothing but having more than one catch block per try block.

The compiler decides to which catch block should be executed if the exception mentioned in the Catch block and the actual exception raised are matches.

We can write an n-number of catch blocks, but make sure that you are writing the exceptions from narrow to broad. What I mean is we should write the specific exception in the first catch and little less specific or parent of specific exception in second catch block so on.

Below the pyramid or narrow to the broad example of **NoSuchElementException**



```
try{
   // no element is present with such id
   driver.findElement(By.id("this-id-isnot-present")).click();

}catch(NoSuchElementException e){
   System.out.println("element is not there");
}catch (NotFoundException e) {
   System.out.println("parent of specific");
}catch (WebDriverException e) {
   System.out.println("broader then parent");
}catch (Exception e) {
   System.out.println("top most exception class");
}
```

Above code deal with NoSuchElementException and its parents, but we can also add random exception which does not have any relationship with other exceptions.

```
try{
   // no element is present with such id
   driver.findElement(By.id("this-id-isnot-present")).click();


}catch (InvalidElementStateException e) {
   System.out.println("element present but not enabled");
}catch(NoSuchElementException e){
   System.out.println("element is not there");
}
```

Combining multiple exceptions in One Catch block :

If we know, a particular block of code is going to throw exceptions, i.e., when you are not sure which exception among few exceptions, and if those exceptions have any common Parent, then we can use the parent exception class in the catch block to handle the scenario.

But when you know that a particular code is going to throw exceptions which are not related at all, then we will not have any common parent class, So in this case, we have to use multiple exceptions in the same catch block using | (pipe) operator.

Syntax : catch(Exception1 | Exception2 | Exception3.... e)

```
try{
   // no element is present with such id
   driver.findElement(By.id("this-id-isnot-present")).click();
   // sleep method throws InterruptedException exception
   Thread.sleep(1000);
}catch (InvalidElementStateException | InterruptedException e) {
   System.out.println("element present but not enabled");
}
```

Basic rules for catch block :

1. Only one catch block will be executed per exception, and only one exception can raise at a time
2. We should order the catch block from narrow type exception to broad.
3. We cannot have more than one catch block with a particular exception if we try to do so the second catch block will give an error saying 'not reachable code.'

More than Two Browser Windows

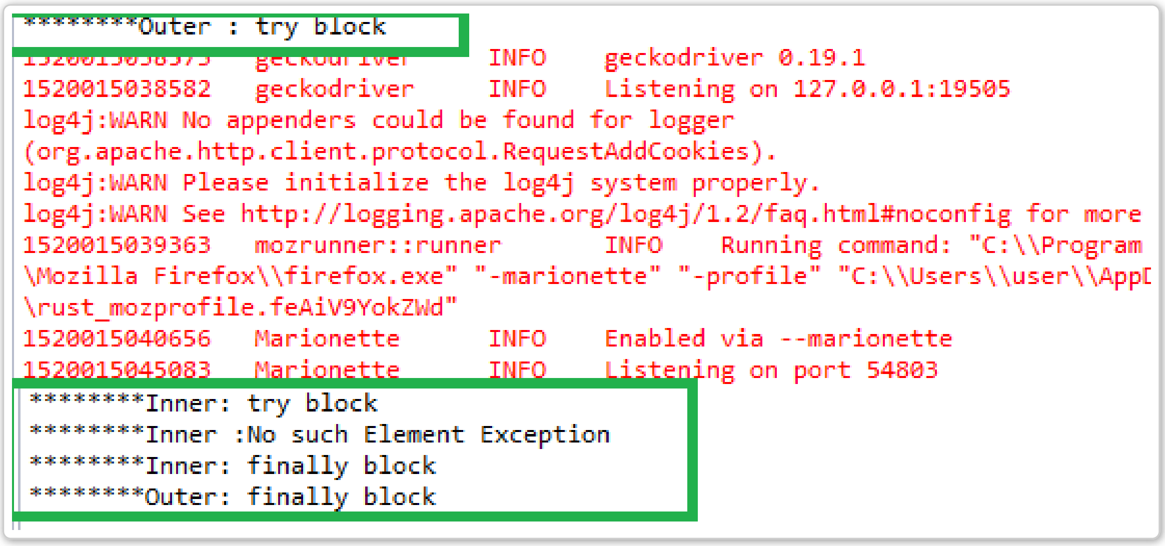
Nested try..catch..finally blocks

When a try..catch block is present inside another try..catch block then it is called as nested try..catch. We can write a try..catch..finally inside a try block of code.

In the below code, inner try block throws **NoSuchElementException**, and we have a catch block with the same exception to handle the exception, then finally block of the inner try will be executed.

Once inner try..catch..finally block is over, the control comes to outside catch block as there is no exception is remaining in an outer try block so catch block will not be executed, but finally in the outer catch block will be executed.

```
public class Nested {
   public static void main(String[] args) {
      try{
         System.out.println("********Outer : try block");
         // set the geckodriver.exe property
         System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
         // open firefox
         WebDriver driver = new FirefoxDriver();
         driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
         driver.get("https://chercher.tech/java/index-selenium-webdriver");
         try{
            System.out.println("********Inner: try block");
            // no element is present with such id
            driver.findElement(By.id("this-id-isnot-present")).click();
            // sleep method throws InterruptedException exception
         }catch(NoSuchElementException e){
            System.out.println("********Inner :No such Element Exception");
            }finally {
                System.out.println("********Inner: finally block");
            }
        }catch (Exception e) {
            System.out.println("********Outer: Catch block");
        }finally{
            System.out.println("********Outer: finally block");
        }
    }
}
```



```
********Outer : try block
1520015038373   geckodriver     INFO    geckodriver 0.19.1
1520015038582   geckodriver     INFO    Listening on 127.0.0.1:19505
log4j:WARN No appenders could be found for logger
(org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more
1520015039363   mozrunner::runner       INFO    Running command: "C:\\Program
\Mozilla Firefox\\firefox.exe" "-marionette" "-profile" "C:\\Users\\user\\AppD
\rust_mozprofile.feAiV9YokZWd"
1520015040656   Marionette      INFO    Enabled via --marionette
1520015045083   Marionette      INFO    Listening on port 54803
********Inner: try block
********Inner :No such Element Exception
********Inner: finally block
********Outer: finally block
```

We can write the nested try-catch block inside catch and finally blocks as well.

```
try{
    // no element is present with such id
    driver.findElement(By.id("this-id-isnot-present")).click();

}catch (Exception e) {
    System.out.println("********Outer: Catch block");
    try{
        System.out.println("**********Inner: try block");
        System.out.println("title of the page is : "+ driver.getTitle());
    }catch(NotFoundException e1){
        System.out.println("********inner: catch block");
    }
}finally{
    System.out.println("********Outer: finally block");
        }
```

An outer catch block handles exceptions raised by inner try block :

If any exception occurred in the inner try block will be handled by the inner catch block, but if the inner catch is not able to handle the exception, then the outer catch block tries to handle the exception raised by the inner try block.

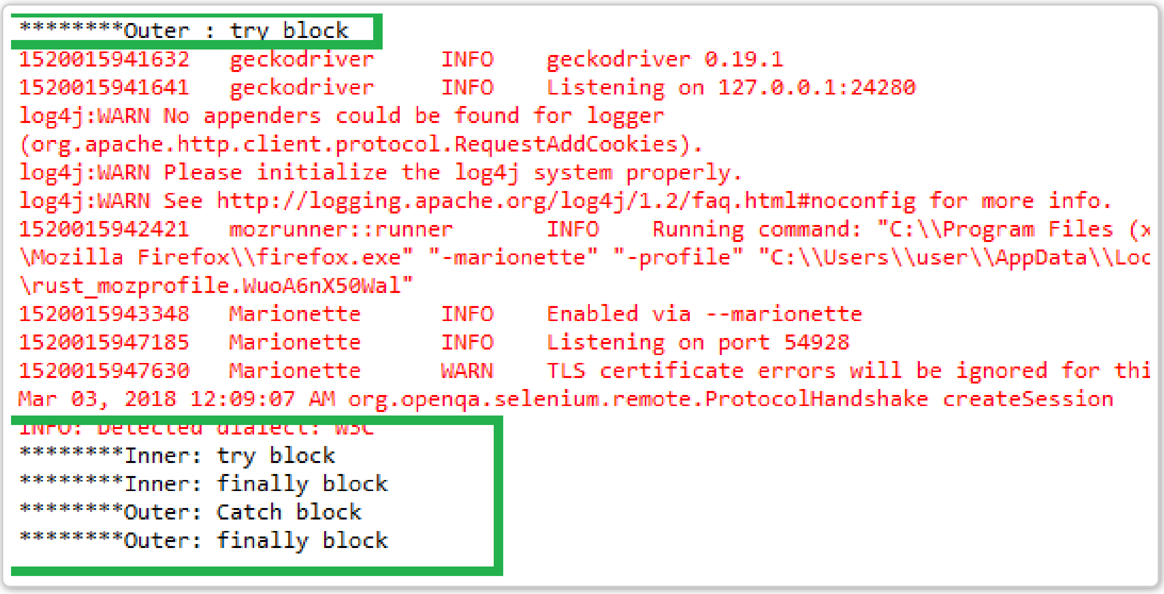
> So whenever we write a nested try-catch block, the inner catch block always will have more than one catch block by default.

In the below program, we will see how the outer works

```
public class TwoCatches {
    public static void main(String[] args) {
        try{
            System.out.println("********Outer : try block");
            // set the geckodriver.exe property
            System.setProperty("webdriver.gecko.driver", "C:/Users/user/Pictures/geckodriver.exe");
            // open firefox
            WebDriver driver = new FirefoxDriver();
            driver.manage().timeouts().implicitlyWait(5, TimeUnit.SECONDS);
            driver.get("https://chercher.tech/java/index-selenium-webdriver");
            try{
                System.out.println("********Inner: try block");
                // no element is present with such id
                driver.findElement(By.id("this-id-isnot-present")).click();
                // sleep method throws InterruptedException exception
            }catch(InvalidElementStateException e){
                System.out.println("********Inner :No such Element Exception");
            }finally {
                System.out.println("********Inner: finally block");
            }
        }catch (NoSuchElementException e) {
            System.out.println("********Outer: Catch block");
        }finally{
            System.out.println("********Outer: finally block");
        }
    }
}
```

In above program, the inner catch block will not be able to handle the exception thrown by the inner try block, so now the exception comes to the outer exception block, and outer exception is able to handle the exception as the exception raised by the inner try block is matching with the exception we mentioned in outer catch block.

From the below image, you can analyze the execution of the above program.



```
********Outer : try block
1520015941632   geckodriver     INFO    geckodriver 0.19.1
1520015941641   geckodriver     INFO    Listening on 127.0.0.1:24280
log4j:WARN No appenders could be found for logger
(org.apache.http.client.protocol.RequestAddCookies).
log4j:WARN Please initialize the log4j system properly.
log4j:WARN See http://logging.apache.org/log4j/1.2/faq.html#noconfig for more info.
1520015942421   mozrunner::runner       INFO    Running command: "C:\\Program Files (x
\Mozilla Firefox\\firefox.exe" "-marionette" "-profile" "C:\\Users\\user\\AppData\\Loc
\rust_mozprofile.WuoA6nX50Wal"
1520015943348   Marionette      INFO    Enabled via --marionette
1520015947185   Marionette      INFO    Listening on port 54928
1520015947630   Marionette      WARN    TLS certificate errors will be ignored for thi
Mar 03, 2018 12:09:07 AM org.openqa.selenium.remote.ProtocolHandshake createSession
INFO: Detected dialect: W3C
********Inner: try block
********Inner: finally block
********Outer: Catch block
********Outer: finally block
```

throws keyword in Exceptions

Throws keyword used in the methods signature, to delegate the *exception to the caller.*

We can handle the exception using try..catch but sometimes there will be a scenario where you should not handle the exception, In such cases, we must delegate the exception using throws keyword.

For example, you are trying to read a file based on the path passed by the user; there is a chance that the file may or may not present in the local system. In this case, you should not handle the exception as you do not know what the file contains in it.

So we must inform the user by delegating the exception, and it is up to the end customer to handle the exception or to stop the program.

```
public class ThrowsKeyword {
   // this method is caller
   public static void main(String[] args) {
      try {
         // exception is handled in caller
         String path = "C:PATH   estFile.properties";
         calledMethod(path);
        } catch (IOException e) {
         e.printStackTrace();
        }
   }

   // this method does not handle the exception instead it delegates the exception
    public static void calledMethod(String filePath) throws IOException{
      // load the properties file
       FileInputStream fis = new FileInputStream(filePath);
       Properties prop = new Properties();
       prop.load(fis);
    }
}
```

Note : We should not try to delegate the exception all the times, sometimes we should handle the exception. We should handle the exception when we have a better solution to overcome the exception.

As per me, all the Checked Exceptions must be delegated to the caller method.

throw keyword in Exceptions

So far, we are trying to catch and handle the exceptions. Before you can catch an exception, some code somewhere must throw one.

Any code can throw an exception: your code, code from a package written by someone else such as the packages that come with the Java platform selenium webdriver, or the Java runtime environment.

Regardless of what/who throws the exception, it's always thrown using a throw statement.

Below the program throws Exception with some reason; we can throw any exception using throw keyword.

```
public class SimpleThrow {
   public static void main(String[] args) throws Exception {
      System.out.println("Before throwing an exception");
      throw new Exception("Bingo,, we are throwing an exception");
   }
}
```

Output :

```
Before throwing an exception
Exception in thread "main" java.lang.Exception: Bingo,, we are throwing
an exception
	at trycatch.SimpleThrow.main(SimpleThrow.java:6)
```

Difference between throw and throws keywords

The throw keyword handover user created an exception to JVM manually, throws keyword is used to delegate the responsibility of exception handling to the caller of the method.

The throw keyword is followed by an exception object; throws keyword is followed by the list of the exception class, which can occur in the method.

The throw keyword can throw only one exception object; the throws keyword can declare multiple exception classes separated by a comma or can have a topmost exception to delegate all the subclass exceptions.

the throw keyword is used in method implementation, but throws is used in the method signature.

A checked exception cannot be propagated using throw only, but a Checked exception can be propagated with throws.

Please refer to the syntaxes in the above topic.

Re-Throw Exception

Re-throw is nothing but throwing an Exception that we have handled

Sometimes we may need to inform the customer that we received an exception by stopping the program

But before stopping, we may need to analyze the reasons, or we may want to perform tasks like disconnecting from the database or closing the browser in selenium. If we do not end a few critical things, we may face issues while reconnecting to the resources.

> Re-throw is not a keyword like a **throw**, re-throw is just a concept which uses **throw**keyword

In below example, we are throwing an exception, and handling it then re-throwing the same exception

```
public static void main(String[] args) throws Exception {

   System.out.println("This is Re-throw example");
   try {
      throw new Exception("Bingo,, we are throwing an exception");
   } catch (Exception e) {
      System.out.println("Exception Reason :: "+ e.getMessage());
      System.out.println("we are gonna rethrow exception");
      // because we re-throwing an exception, we must add throws in method signature
      throw e;
   }
}
```

Output :

rethrow-exception-java-selenium

FFE : Frequently Faced Exception in Java

There are n-number of exceptions present in java, but in this topic, we will discuss the few Frequently faced Exceptions in Java, these exceptions are also frequently faced by selenium webdriver tester as well.

1. ArithmeticException : **ArithmeticException** occurs whenever our math calculations result in infinite.

```
public static void main(String[] args) {
   System.out.println("Results in infinite : "+ 1/0);
}
```

arithmatic-exception-java-selenium

Output :

2. NullPointerException : **NullPointerException** is associated with object creation whenever you try to access the methods inside an object without creating an object.

By default, all the objects will have a null value, if we try to access any method without initiating the object; then, you will face NullPointerException.

Simple words: you get NullPointerException exception when you call any method on a null object.

```
public static void main(String[] args) {
   String str=null;
   System.out.println("String length : "+str.length());
}
```

null-pointer-exception-java-selenium

Output :

3. NumberFormatException : **NumberFormatException** occurs when we try to convert an alphabet into a number using the Integer Wrapper class.

```
public static void main(String[] args) {
   //converting string '10' into number 10
   int ten = Integer.parseInt("10");

   // converting 'abc' into number (throws exception
   int abc = Integer.parseInt("abc");
}
}
```

Output : numberformatexception-java-selenium

ArrayIndexOutOfBoundsException : **ArrayIndexOutOfBoundsException** exception occurs when you try to access a invalid index number in a array.

For Example, you have an array for 4 elements, but if you try to access the 9th element, the system throws ArrayIndexOutOfBoundsException.

```java
public static void main(String[] args) {
   int[] intArray = {10,20,30,40};
   // trying to access 4 element using index 3 as indexes starts from 0
   System.out.println("3rd index (4th element) : "+intArray[3]);
   //access for 9th element
   System.out.println("9th element is : "+intArray[8]);
}
```

Output : array-index-out-of-bounds-exception-java-selenium

MalFormedURLException : **MalFormedURLException** is thrown when the built-in URL class encounters an invalid URL; specifically, when the protocol that is provided is missing or invalid.

The below program throws MalFormedURLException because I have not added HTTP to the URL. URL class expects fully form URLs like **https://chercher.tech**

MalFormedURLException is Checked exception, so we have to use throws with the method, or we should handle the exception using try..catch

```
public static void main(String[] args) throws MalformedURLException {
   URL u1 = new URL("chercher.tech");
}
```

Output : MalformedURLException-java-selenium

ClassNotFoundException : **ClassNotFoundException** As the name suggests ClassNotFoundException in Java is a subclass of java.lang.Exception and Comes when Java Virtual Machine tries to load a particular class and doesn't found the requested class in classpath. You need to add the required jar in your classpath.

NoClassDefFoundError : **NoClassDefFoundError This is** caused when there is a class file that your code depends on, and it is present at compile time but not found at runtime. Look for differences in your build time and runtime classpaths.

IllegalArgumentException : **IllegalArgumentException Thrown when** a method receives an argument formatted differently than the method expects. The only thing you must do is correct the values of the input parameters.

ClassCastException : **ClassCastException** Thrown to indicate that the code has attempted to cast an object to a subclass of which it is not an instance.

```
Object i = Integer.valueOf(100);
String s = (String)i;
```

Solution: Be careful when trying to cast an object of a class into another class. Ensure that the new type belongs to one of its parent classes.

You can prevent the ClassCastException by using Generics because Generics provide compile-time checks and can be used to develop type-safe applications.

IllegalStateException : **IllegalStateException** It is used to indicate that “a method has been invoked at an illegal or inappropriate time.

Custom Exception

If you are creating your own Exceptions that are known as Custom Exception or User-defined Exception. Java Custom Exception’s are used to customize the exception according to user needs.

Steps to Create Custom Exception :

Create a java class ThisIsCustomDumbException, as best practice create custom exception class name ending with 'Exception'

Create a constructor for the ThisIsCustomDumbException with accepting an error message as a parameter.

Call the superclass constructor using a super() keyword; the superclass is nothing but Exception class.

If you want to perform something, then you can perform in the constructor or by creating other methods and giving them a call from the constructor. We are done creating a custom exception.

```
public class ThisIsCustomDumbException extends Exception{
   // constructor of this class
   public ThisIsCustomDumbException(String errorMessage) {
      // we have to pass message to the super class constructor
      super(errorMessage);
   }
}
```

Steps to Use Custom Exception :

1. Create java class 'UseCustomException'
2. Create the main method
3. Throw our custom exception using throw keyword.
4. The system expects us either put the throw exception inside try..catch or add throws keyword with an Exception class

```
public class UseCustomException {
   public static void main(String[] args) throws ThisIsCustomDumbException {
      throw new ThisIsCustomDumbException("Author is dumb guy");

   }
}
```

The output of custom Exception:

custom-exception-selenium-java

Exceptions in Selenium Webdriver

There are 40 exceptions present in selenium webdriver, but most of the time, we face only 10-15 exceptions; this tutorial will explain all the exceptions present in the Selenium.

This tutorial's scope is to discuss, what are the exception present in the selenium, when we get them, and how to mitigate them.

The architecture of Webdriver Exceptions

webdriver-exception-architecture

Selenium Webdriver Exceptions

WebDriverException :

WebDriverException is the top most exception class in the selenium webdriver, and it extends RuntimeException class. All selenium Exceptions are subclasses of WebDriverException class.

selenium Exceptions are subclasses of WebDriverException class.

NotFoundException :

NotFoundException is a subclass of WebDriverException, NotFoundException occurs when a thing on the DOM is not present; we may not directly receive this exception, but sub-classes of these exceptions occur during execution.

Let's discuss the Exceptions present in the NotFoundException class, then we will learn about WebDriverException.

not-found-exception-selenium-webdriver

In this tutorial, I have given a sample code that causes the exception, that does not mean only this piece of code will raise an exception. Other commands also can raise exceptions.

NoSuchElementException :

Webdriver throws NoSuchElementException when there is no matching element present on the webpage.

Reasons : NoSuchElementException occurs because of one of the below reasons

*Either **XPath** you specified for the element is wrong.*
The second reason is AJAX has not returned yet, and you've already obtained NoSuchElementException
The page is still being rendered, but you've already finished your element search because of a low wait time.
The element is not on the page at all.

Code which causes the NoSuchElementException :

```
driver.findElement(By.xpath("//label/span"))
```

> Exception Message:org.openqa.selenium.NoSuchElementException: Unable to locate element: //label/span

NoSuchWindowException :

selenium webdriver throws NoSuchWindowException when the user tries to switch to or perform an **operation on browser windows** when the window is not present.

Reasons : NoSuchWindowException occurs because of one of the below reasons

1. Current window closed
2. The target window is not opened yet, but you were trying to perform the operation
3. When the target window has been closed, but still the code tried to perform an operation on the target window.

Code which causes the NoSuchWindowException :

```
driver.switchTo().window("window Gu ID");
```

NoSuchFrameException :

NoSuchFrameException happens when the user tries to switch to a frame that is not present at the time.

Reasons : NoSuchFrameException occurs because of one of the below reasons

1. There is no **frame with given details**, or the details changed during the page loading time
2. Target Frame is not actually a frame if the locators match exactly with another element(as the first instance).
3. We have not given enough wait time to load the frame.

Code which causes the NoSuchFrameException :

```
driver.switchTo().frame("frame1");
```

NoAlertPresentException :
Selenium throws NoAlertPresentException exception when the user tries to access an alert popup, which is currently not present.

Here Alert popup could be **Alert Box, Confirmation Box, Prompt Box** from the Javascript commands.

Reasons : NoAlertPresentException occurs because of one of the below reasons

When a user tries to access alert which is not present in the at the current time

When javascript is blocked in the browser

Not giving enough time for the alert to load.

When an alert is already closed

Code which causes the NoAlertPresentException :

```
driver.switchTo().alert()
```

StaleElementReferenceException :

TimeoutException :

TimeoutException is related with **explicitwait, fluentwait**, pageLoadTimeOut, scriptLoadTimeout. Selenium webdriver throws the TimeoutException exception when the user has defined the wait time, but the elements have not been found within the wait time.

Reasons : TimeoutException occurs because of one of the below reasons

Element is not present or given Condition is not met.

When the given timeout is very low

When the page takes more time to load

When executeAsyncScript() takes more time to return its operation

Code which causes the TimeoutException :

```
WebDriverWait wait = new WebDriverWait(driver, 30);
WebElement idElement;
idElement = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("123")));
```

ScreenshotException :

ScreenshotException exception will be thrown when selenium fails to **take a screenshot of the webpage**. If ScreenshotException occurs, then the screenshot captured turns black.

Reasons :

1. When we try to **take a screenshot** in the **headless browser**(phantomjs, htmlunitdriver) without enabling the screenshot functionality.

Code which causes the ScreenshotException :

```
TakesScreenshot screenShot =((TakesScreenshot) driver);
```

ScriptTimeoutException :

webdriver throws ScriptTimeoutException exception when executeAsyncScript takes more time than a given time limit to return the value.

JavascriptException :

UnhandledAlertException :

selenium throws UnhandledAlertException exception when the user tries to perform while alert(javascript popup) is present on the page. Javascript pop-ups will not allow performing manual actions(manual testing) on the page when it is present on the page.

Reasons :

1. Javascript Popup is present on the page
2. You have written a code to accept the pop up automatically, but still, you are trying to handle the pop-up.
3. You might have missed code to handle the pop-up.

Insight : Users can use the getAlertText method to get the text from the alert when UnhandledAlertException exception occurs.

Compare Screenshots in selenium

UnexpectedTagNameException :

UnexpectedTagNameException occurs when the user is using Select Class in selenium to perform an action on the dropdown/element where the element is not formed with a **select** HTML tag.

Reasons :

1. The dropdown is formed without using a **select** tag name.
2. Html code might have changed to a non-select tag after writing the test automation code.
3. If dropdown HTML code is specific to the browser used, Sometimes code can be rendered based on the browser.

Code which causes the UnexpectedTagNameException :

```
// throws the UnexpectedTagNameException
Select dropdown = new Select( driver.findElement(By.id("//input"))

// this will not throw exception
Select dropdown = new Select( driver.findElement(By.id("//select")) );
```

UnReachableBrowserException :

UnReachableBrowserException will be thrown when the selenium webdriver is not able to connect to the browser after starting the browser.

Reasons :

Browser is taking more time to invoke after the object creation

Browser died in between test execution if you run the large test suite.

The browser is not opened in virtual machines or the remote driver.

The provided server address to RemoteWebDriver is invalid, so the connection could not be established.

The browser version and the driver server.exe / selenium version are not compatible.

When you use IEdriver64.exe instead of IEdriver32.exe when the system is 32-bit architecture.

Note : IEDriver32.exe can be used in place of IEdriver64.exe, but vice versa is not possible

NoSuchSessionException :

Selenium throws this exception when selenium is not able to connect to the browser session; this exception occurs mostly on Chromedriver

Reasons :

When the temp memory is running out of space whenever we run on chrome; the chrome driver server occupies and leaves the cache file on the temp directory

This exception occurs when you run on Docker mostly (We run our tests inside a docker container, and the docker default dev/shm size is 64mb. Increasing this resolved the "no such session" issue for us. We use docker-compose, so just added shm_size: 256M to the docker-compose.yml file.)

Browser and Driver server.exe file version mismatch

The problem which appears when running tests for a prolonged period

MoveTargetOutOfBoundsException :

Selenium throws MoveTargetOutOfBoundsException when the user tries to move/drag the element or the cursor outside the visible screen.

Reasons :

1. Change of screen size (test cases are written in desktop, but current execution is happening laptop, considering that desktop screen size is bigger)

Code which causes the MoveTargetOutOfBoundsException :

```
Actions actions = new Actions(driver);
actions.mouseMove( 10, 25 )
actions.perform();
```

InvalidElementStateException :

Selenium Webdriver throws InvalidElementStateException when user tries to perform operation on a Webelement whose state is disabled.

Subclasses of this Exception gives detail explanations, the subclasses are **ElementNotInteractableException, ElementNotSelectableException**

Reasons :

1. When element state is disabled

Code which causes the InvalidElementStateException :

```
<input type='button' disabled=true value='save'>

// click the button which is disabled
driver.findElement(By.xpath("//input[@type='button']")).click();
```

invalid-element-state-exception-selenium-webdriver

ElementNotInteractableException :

ElementNotInteractableException is throws when we try to perform operation on an element which not intereactable at the moment. ElementNotInteractableException has two subclasses which are ElementNotVisibleException, ElementClickInterceptedException.

Reasons :

1. When element size is 0 x 0 (width =0 and height =0)

Code which causes the ElementNotInteractableException :

```
<input type='button' height=0px width=0px value='save'>

// click the button which has width=0 and height=0
driver.findElement(By.xpath("//input[@type='button']")).click();
```

element-not-intereactable-exception-selenium-webdriver

ElementNotVisibleException :

Selenium throws ElementNotVisibleException when the user tries to perform an operation on a web element, which is present on the page, but the element is currently not visible.

Reasons :

1. Clicking an element which is not visible
2. **sendkeys** to element which is not visible

3. The element may not be visible because there are more elements under different pages, but that page source code is still present in the cache.

Code which causes the ElementNotVisibleException :

```
<input type='button' hidden=true value='save'>

// click the button which is hidden (not visible)
driver.findElement(By.xpath("//input[@type='button']")).click();
```

ElementClickInterceptedException :

Indicates that a click could not be properly executed because the target element was obscured in some way; I never faced this exception in my testing life.

ElementNotSelectableException :

ElementNotSelectableException is related with Select in selenium, when user tries o select a option which is not selectable..

UnSupportedCommandException :

Webdriver has methods that give a call to native methods present in the browser, and the browser reacts to the native commands. In recent times we have received updates for a driver like geckodriver.exe and chromedriver.exe.

These drivers do not support some selenium commands, and if we try to perform those commands using selenium then those go through the driver servers, and selenium throws unSupportedCommandException if the driver server does not support the command
Examples : Gecko driver does not support Actions class

InvalidCoordinatesException :

Selenium throws InvalidCoordinatesException when a user tries to move the mouse to a co-ordinate, which is not valid using action class methods.
Reasons :

1. When we run our test in different sizes of monitors, we may face this issue.

ImeNotAvailableException :

It 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 :

This indicates that activating an IME engine has failed.

ConnectionClosedException :

Selenium throws ConnectionClosedException when a program tries to perform an operation on the safari browser but where the safari browser is disconnected from the selenium session.

ErrorHandler.UnknownServerException :

Exception used as a place holder if the server returns an error without a stack trace.

InvalidCookieDomainException :

webdriver throws InvalidCookieDomainException when the user attempts to add a cookie under a different domain (URL) than the current URL.

JsonException :

Session capabilities normally return the values in JSON and selenium parses it, JsonException occurs when a user tries to get the session capabilities where the session is not created.

UnableToCreateProfileException :

We can open a browser with certain options using profiles, sometimes a new version of selenium driverserver or browser version may not support the profiles, during such cases UnableToCreateProfileException raises.

UnableToSetCookieException :

Webdriver throws UnableToSetCookieException when a driver fails to set a cookie.

Recommended Readings

Listeners in Selenium

TestNG unit Testing Framework in Selenium

TestNG @Annotations in Selenium

@Test Attributes in TestNG with Selenium

Cookies in Selenium Webdriver [Latest]

Take Screenshot in selenium | Element Screenshot | Store PDF | Highlight

Multiple Windows Handling in Selenium

iFrames / Frames in Selenium