# 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, exception stops the execution, but we are more powerful than 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 more meaningful message without stopping the execution of the program.

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

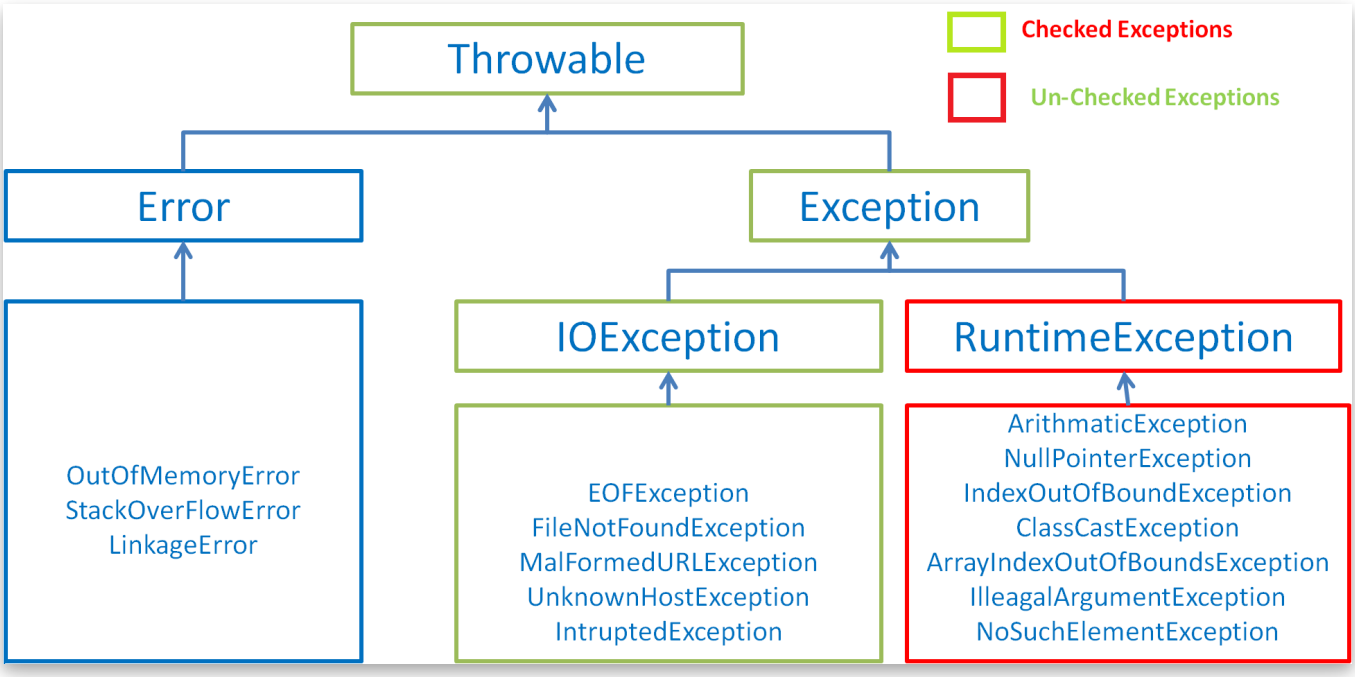
```
public class JavaException {
 public static void main(String[] args) {
  System.out.println("Before Exception");
  // below code throws Exception
  Integer intValue = new Integer("chercher tech");
  System.out.println("Converted value : " + intValue);
  System.out.println("After Exception");
 }
}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

In above program the line *Integer("chercher tech")* tries to convert the alphabet string into number. Converting the alphabets into 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 handle the error. Errors are mainly caused by the environment in which application is running. Errors are not checked at compile time and do not have to be (but can be) caught or handled.

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

*OutOfMemoryError*

*StackOverFlowError*

*LinkageError*

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

*Checked Exceptions also 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 a 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 donot do either of them, then 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 occurs during the run time of the program. Compiler will not be able to detect the unchecked exception during compile time.

Below are few unchecked exceptions :

*ArithmaticException*
*NullPointerException*
*IndexOutOfBoundsException*
*ClassCastException*
*ArrayIndexOutOfBoundsException*
*NumberFormatException*

## Difference between Error and Exception

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

User should never design an application to throw Error, but 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, compiler will not have knowledge of it. Checked exceptions are known to compiler where as 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 application is running. Exceptions are mainly caused by the application itself.

StackOverflowError, OutOfMemoryError are few examples for Error, IOException, NullpointerException are 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 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 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 methods fetches the detailed exception details like which line caused the exception, which all are methods got affected, what is the error message and few more details.

5. printStackTrace() : This methods 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 webpage, program throws an exception when we try to access a file which 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 certain 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 hospital which is little sensitive and we are having 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 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 be having code which will efficiently handle that particular exceptional situation

In catch block user should mention what is the exception that we should handle because we cannot handle all the

exceptions as different exceptions will require a different code to handle the situation.

Catch block will not be executed if there is no exception occurs in 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
```

Above program throws 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 occurred or not in 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 try block

## When finally block will not execute :

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

*When JVM crashes*

*When try block is executed infinitely without breaking*

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

In 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*

```
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 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 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 above image you can see that finally and block executed, and there is not 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*

*catch block is optional when finally block is present*

*finally block is optional when 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..finaly

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

1. Open 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 didnot close the browser i.e as there no element with id='this-id-isnot-present', so find element throws 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();
 }
}
```

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 answer enter your answer in comment section of the page

## 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 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 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 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 element is there on the webpage or not, but we get conclusion that element is not present even though the element is present.

This is the reason why we should write specific exception in 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"); }
```

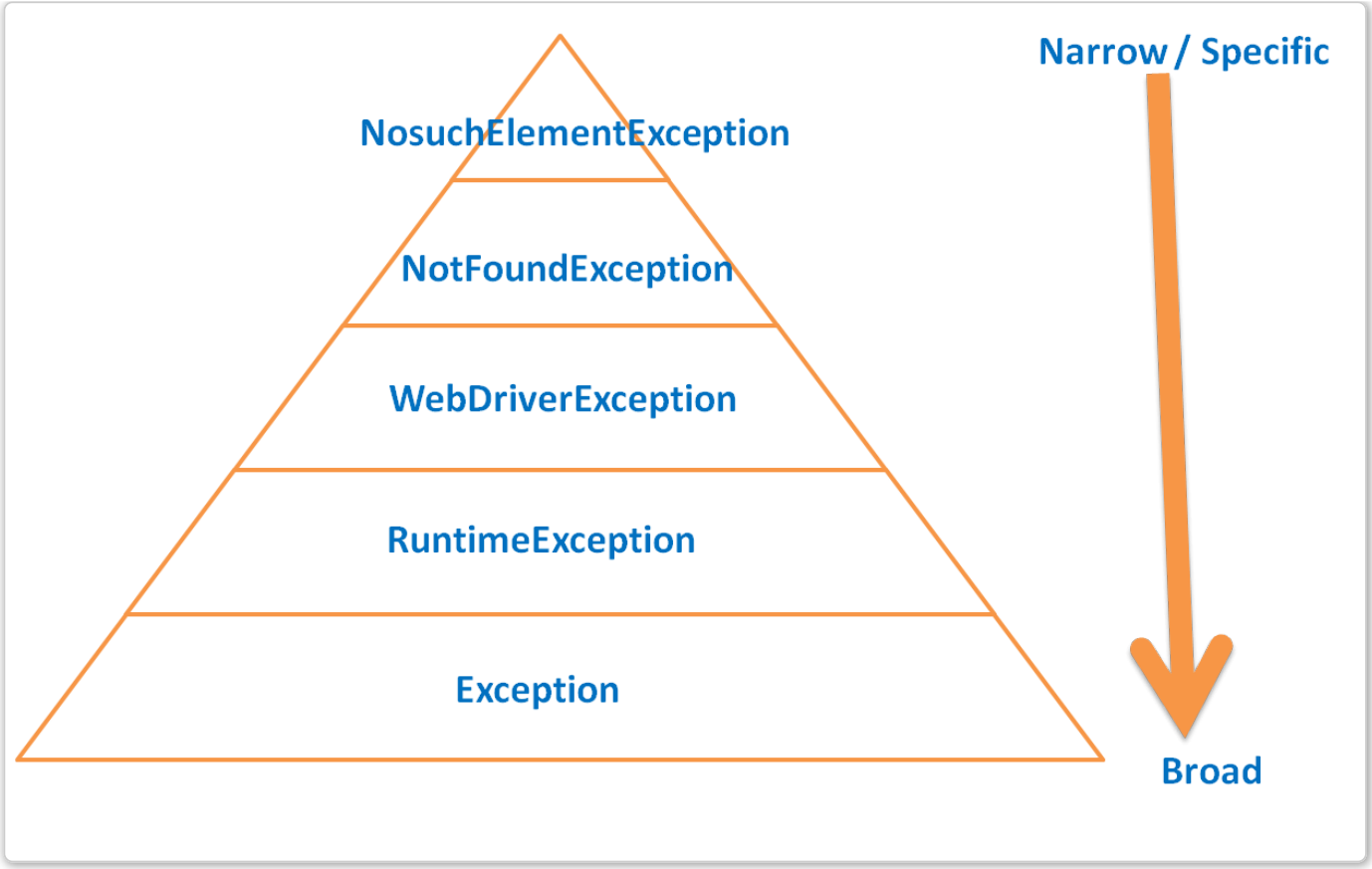
## Multiple catch blocks with try block

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

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

We can write 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 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 **donot 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 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 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 :*

*Only one catch block will be executed per exception and only one exception can raise at a time*

*We should order the catch block from narrow type exception to broad.*

*We cannot have more than one catch block with a particular exception, if we try to do so the second catch block will give error saying 'not reachable code'*

## Nested try..catch..finally blocks

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

**In below code inner try block throws 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 outer try block so catch block will not be executed, but finally in 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
1520015038573   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"); }**

**Outer catch block handles exceptions raised by inner try block :**

**If any exception occurred in inner try block will be handled by the inner catch block, but if 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 nested try catch block the inner catch block always will have more than one catch block by default.

In 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

```
driver.findElement(By.id("this-id-is-not-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 below image you can analyze the execution of 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 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 chance that the file may or may not present in the local system. In this case you should not handle the exception as you donot know what the file contains in it.

So we must inform the user by delegating the exception, and it is upto end customer to handle 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 better solution to overcome the exception.

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

## throw keyword in Exception

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 throw statement.

Below 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"); } }

```
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 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 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 <g class="gr_ gr_484 gr-alert gr_gramm gr_