# Handling Exceptions Using Exception Handling Framework in Selenium Scripts

We will learn about **types of exceptions and how to handle exceptions in java and Selenium scripts**. Developers/testers use exception handling framework to handle exception in selenium scripts.

**Example**: When selenium script fails due to wrong locator, then developer should be able to understand the reason for failure and this can be achieved easily if the exception is handled properly in the program.

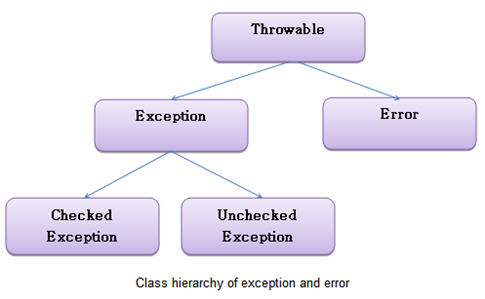
Below we have described the types of exceptions and the different ways how we can use exception handling framework in selenium scripts.

Exceptions are events due to which java program ends abruptly without giving expected output. Java provides a framework where user can handle exceptions.

**There are three kinds of exceptions:**

1. Checked Exception
2. Unchecked Exception
3. Error

**Class hierarchy of exception and error:**

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

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

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

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

**Example**: *ArrayIndexoutOfBoundException*

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

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

**Exception handling:**

**Try and Catch block:**

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

**Example**:

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

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

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

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

**Example**:

------------

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

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

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

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

***throws* Exception:**

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

**Example**:

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

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

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

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

***finally* block:**

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

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

**Example**:

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

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

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

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

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

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

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

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

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

**Example**:

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

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

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

**Exceptions in Selenium WebDriver:**

Selenium has its own set of exceptions. While developing selenium scripts, programmer has to handle or throw those exceptions. Below are few examples of exceptions in selenium.

**Examples**:

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

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

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

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

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

**Conclusion:**

Exception handling is the essential part of every java program as well as selenium script. We can build robust and optimal code by**handling exception in smart ways**. And it is also a best practice to handle exceptions in script which will give you a better report when program fails due to any reason.

Here we have tried to cover the process and framework of exception handling which is required to be implemented in selenium scripts.

Remember it is not mandatory to always handle exception in *try-catch* block. You can also throw an exception depending upon the requirement in script.