Exception Handling

In this tutorial post, we will study the concept of Exception Handling in Java and how it works.

The **exception handling in java** is one of the powerful *mechanism to handle the****runtime****errors* so that normal flow of the application can be maintained.

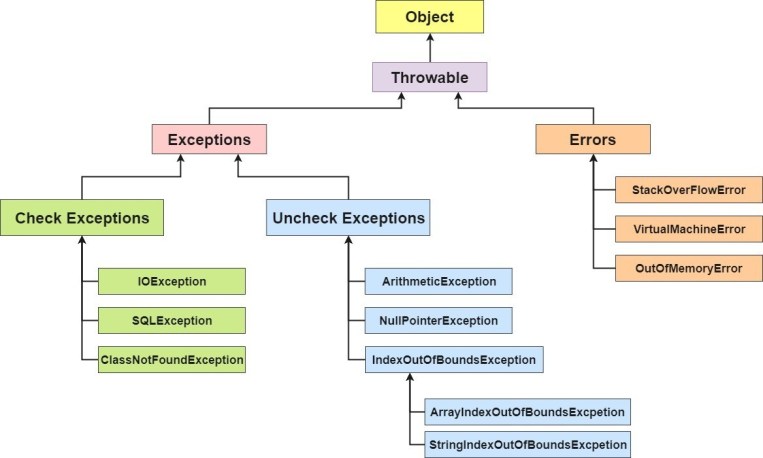
##### **What is an Exception?**

An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e at run time, that disrupts the normal flow of the program’s instructions. It is an **object** which is thrown at **runtime**.

##### **What is exception handling?**

Exception Handling is a mechanism to handle runtime errors such as ClassNotFound, IO, SQL, Remote etc. So it is a way to provide a proper structure when an exception occurs such that the program execution is not affected. Thus the main advantage of exception handling in java is **to maintain the normal flow of the application**.

##### **Hierarchy of Java Exception classes**



All exception and errors types are sub classes of class **Throwable**, which is base class of hierarchy.One branch is headed by **Exception**. This class is used for exceptional conditions that user programs should catch. NullPointerException is an example of such an exception.Another branch,**Error** are used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). StackOverflowError is an example of such an error.

There are basically 3 types of exceptions/abnormal conditions –

* **Checked Exceptions**
* **Unchecked Exceptions**
* **Errors**

###### **1) Checked Exception**

The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

###### **2) Unchecked Exception**

The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

###### **3) Error**

Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

##### **Some Common Scenarios of Exceptions –**

###### **1) Scenario where ArithmeticException occurs**

If we divide any number by zero, there occurs an ArithmeticException.

int a=50/0;//ArithmeticException

###### **2) Scenario where NullPointerException occurs**

If we have null value in any variable, performing any operation by the variable occurs an NullPointerException.

String s=null;

System.out.println(s.length());//NullPointerException

###### **3) Scenario where NumberFormatException occurs**

The wrong formatting of any value, may occur NumberFormatException. Suppose I have a string variable that have characters, converting this variable into digit will occur NumberFormatException.

String s="abc";

int i=Integer.parseInt(s);//NumberFormatException

###### **4) Scenario where ArrayIndexOutOfBoundsException occurs**

If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException as shown below:

int a[]=new int[5];

a[10]=50; //ArrayIndexOutOfBoundsException

|  |  |
| --- | --- |
| ERRORS | EXCEPTIONS |
| Recovering from Error is not possible. | We can recover from exceptions by either using try-catch block or throwing exceptions back to caller. |
| All errors in java are unchecked type. | Exceptions include both checked as well as unchecked type. |
| Errors are mostly caused by the environment in which program is running. | Program itself is responsible for causing exceptions. |
| Errors occur at runtime and not known to the compiler. | All exceptions occurs at runtime but checked exceptions are known to compiler while unchecked are not. |
| They are defined in java.lang.Error package. | They are defined in java.lang.Exception package |
| Examples : java.lang.StackOverflowError, java.lang.OutOfMemoryError | Examples : Checked Exceptions : SQLException, IOException Unchecked Exceptions : ArrayIndexOutOfBoundException, NullPointerException, ArithmeticException. |

##### **Java Exception Handling Keywords –**

There are 5 keywords used in java exception handling.

* ***try***
* ***catch***
* ***finally***
* ***throw***
* ***throws***

##### **Java try block**

Java try block is used to enclose the code that might throw an exception. It must be used within the method. Java try block must be followed by either catch or finally block or both..

###### **Syntax of java try-catch block**

try{

//code that may throw exception

}catch(Exception\_class\_Name ref){

}

###### **Syntax of try-finally block**

try{

//code that may throw exception

}finally{}

##### **Java catch block**

Java catch block is used to handle the Exception. It must be used after the try block only. You can use multiple catch block with a single try.

##### **Java finally block**

**Java finally block** is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block follows try or catch block.

##### **Problem without exception handling**

Let’s try to understand the problem if we don’t use try-catch block.

public class Testtrycatch1{

public static void main(String args[]){

int data=50/0;//may throw exception

System.out.println("rest of the code...");

}

}

o/p

Exception in thread main java.lang.ArithmeticException:/ by zero

As displayed in the above example, rest of the code is not executed (in such case, rest of the code… statement is not printed). There can be 100 lines of code after exception. So all the code after exception will not be executed.

##### **Solution by exception handling**

public class Testtrycatch2{

public static void main(String args[]){

try{

int data=50/0;

}catch(ArithmeticException e){System.out.println(e);}

System.out.println("rest of the code...");

} }

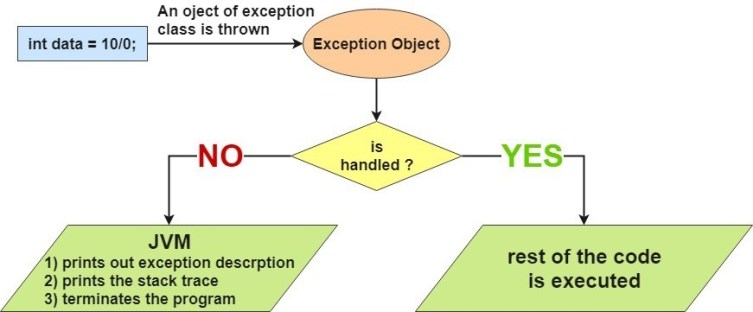
OP

Exception in thread main java.lang.ArithmeticException:/ by zero

rest of the code...

Now, as displayed in the above example, rest of the code is executed i.e. rest of the code… statement is printed.

##### **Internal working of java try-catch block**



The JVM firstly checks whether the exception is handled or not. If exception is not handled, JVM provides a default exception handler that performs the following tasks:

* Prints out exception description.
* Prints the stack trace (Hierarchy of methods where the exception occurred).
* Causes the program to terminate.

But if exception is handled by the application programmer, normal flow of the application is maintained i.e. rest of the code is executed.

##### **Program Example of Try Catch & Finally in Java –**

class TryCatchFinallyExample

{

public static void main (String[] args)

{

// array of size 4.

int[] arr = new int[4];

try

{

int i = arr[4];

// this statement will never execute

// as exception is raised by above statement

System.out.println("Inside try block");

}

catch(ArrayIndexOutOfBoundsException ex)

{

System.out.println("Exception caught in catch block");

}

finally

{

System.out.println("finally block executed");

}

// rest program will be executed

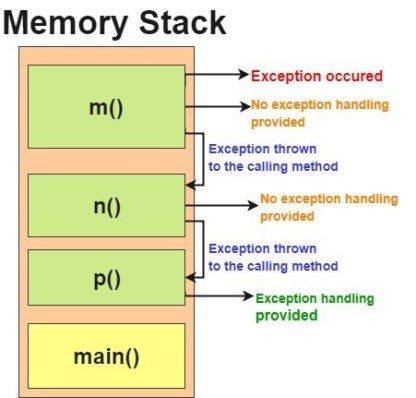
System.out.println("Outside try-catch-finally clause");

}

}

**Output**

Exception caught in catch block  
finally block executed  
Outside try-catch-finally clause



##### **Java throw keyword**

The Java throw keyword is used to explicitly throw an exception. We can throw either checked or uncheked exception in java by throw keyword. The throw keyword is mainly used to throw**custom exception**(we will study about them in further tutorials)

The syntax of java throw keyword is given below.

Syntax:

throw exception;

Let’s see the example of throw IOException.

throw new IOException("sorry device error);

Example

public class TestThrow1{

static void validate(int age){

if(age<18)

throw new ArithmeticException("not valid");

else

System.out.println("welcome to vote");

}

public static void main(String args[]){

validate(13);

System.out.println("rest of the code...");

}

}

O/p

Exception in thread main java.lang.ArithmeticException:not valid

##### **Java throws keyword**

The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmers fault that he is not performing check up before the code being used.

Primary advantage of throws keyword is ***it provides information to the caller of the method about the exception.***

###### **Syntax of java throws**

return\_type method\_name() throws exception\_class\_name{

//method code

}

###### **Java throws example**

Let’s see the example of java throws clause which describes that checked exceptions can be propagated by throws keyword.

import java.io.IOException;

class Testthrows1{

void m()throws IOException{

throw new IOException("device error");//checked exception

}

void n()throws IOException{

m();

}

void p(){

try{

n();

}catch(Exception e){System.out.println("exception handled");}

}

public static void main(String args[]){

Testthrows1 obj=new Testthrows1();

obj.p();

System.out.println("normal flow...");

}

}

**Output**

exception handled

normal flow...

##### **Difference between throw and throws in Java**

There are many differences between throw and throws keywords. A list of differences between throw and throws are given below:

|  |  |  |
| --- | --- | --- |
| **No.** | **Throw** | **throws** |
| 1) | Java throw keyword is used to explicitly throw an exception. | Java throws keyword is used to declare an exception. |
| 2) | Throw is followed by an instance. | Throws is followed by class. |
| 3) | Throw is used within the method. | Throws is used with the method signature. |
| 4) | You cannot throw multiple exceptions. | You can declare multiple exceptions e.g. public void method()throws IOException,SQLException. |

Selenium Exceptions

https://seleniumhq.github.io/selenium/docs/api/py/common/selenium.common.exceptions.html

Exceptions that may happen in all the webdriver code.

exception selenium.common.exceptions.ElementClickInterceptedException(msg=None, screen=None, stacktrace=None)

The Element Click command could not be completed because the element receiving the events is obscuring the element that was requested clicked.

exception selenium.common.exceptions.ElementNotInteractableException(msg=None, screen=None, stacktrace=None)

Thrown when an element is present in the DOM but interactions with that element will hit another element do to paint order

exception selenium.common.exceptions.ElementNotSelectableException(msg=None, screen=None, stacktrace=None)

Thrown when trying to select an unselectable element.

For example, selecting a ‘script’ element.

exception selenium.common.exceptions.ElementNotVisibleException(msg=None, screen=None, stacktrace=None)

Thrown when an element is present on the DOM, but it is not visible, and so is not able to be interacted with.

Most commonly encountered when trying to click or read text of an element that is hidden from view.

exception selenium.common.exceptions.ErrorInResponseException(response, msg)

Thrown when an error has occurred on the server side.

This may happen when communicating with the firefox extension or the remote driver server.

exception selenium.common.exceptions.ImeActivationFailedException(msg=None, screen=None, stacktrace=None)

Thrown when activating an IME engine has failed.

exception selenium.common.exceptions.ImeNotAvailableException(msg=None, screen=None, stacktrace=None)

Thrown when IME support is not available. This exception is thrown for every IME-related method call if IME support is not available on the machine.

exception selenium.common.exceptions.InsecureCertificateException(msg=None, screen=None, stacktrace=None)

Navigation caused the user agent to hit a certificate warning, which is usually the result of an expired or invalid TLS certificate.

exception selenium.common.exceptions.InvalidArgumentException(msg=None, screen=None, stacktrace=None)

The arguments passed to a command are either invalid or malformed.

exception selenium.common.exceptions.InvalidCookieDomainException(msg=None, screen=None, stacktrace=None)

Thrown when attempting to add a cookie under a different domain than the current URL.

exception selenium.common.exceptions.InvalidCoordinatesException(msg=None, screen=None, stacktrace=None)

The coordinates provided to an interactions operation are invalid.

exception selenium.common.exceptions.InvalidElementStateException(msg=None, screen=None, stacktrace=None)

Thrown when a command could not be completed because the element is in an invalid state.

This can be caused by attempting to clear an element that isn’t both editable and resettable.

exception selenium.common.exceptions.InvalidSelectorException(msg=None, screen=None, stacktrace=None)

Thrown when the selector which is used to find an element does not return a WebElement. Currently this only happens when the selector is an xpath expression and it is either syntactically invalid (i.e. it is not a xpath expression) or the expression does not select WebElements (e.g. “count(//input)”).

exception selenium.common.exceptions.InvalidSessionIdException(msg=None, screen=None, stacktrace=None)

Occurs if the given session id is not in the list of active sessions, meaning the session either does not exist or that it’s not active.

exception selenium.common.exceptions.InvalidSwitchToTargetException(msg=None, screen=None, stacktrace=None)

Thrown when frame or window target to be switched doesn’t exist.

exception selenium.common.exceptions.JavascriptException(msg=None, screen=None, stacktrace=None)

An error occurred while executing JavaScript supplied by the user.

exception selenium.common.exceptions.MoveTargetOutOfBoundsException(msg=None, screen=None, stacktrace=None)

Thrown when the target provided to the ActionsChains move() method is invalid, i.e. out of document.

exception selenium.common.exceptions.NoAlertPresentException(msg=None, screen=None, stacktrace=None)

Thrown when switching to no presented alert.

This can be caused by calling an operation on the Alert() class when an alert is not yet on the screen.

exception selenium.common.exceptions.NoSuchAttributeException(msg=None, screen=None, stacktrace=None)

Thrown when the attribute of element could not be found.

You may want to check if the attribute exists in the particular browser you are testing against. Some browsers may have different property names for the same property. (IE8’s .innerText vs. Firefox .textContent)

exception selenium.common.exceptions.NoSuchCookieException(msg=None, screen=None, stacktrace=None)

No cookie matching the given path name was found amongst the associated cookies of the current browsing context’s active document.

exception selenium.common.exceptions.NoSuchElementException(msg=None, screen=None, stacktrace=None)

Thrown when element could not be found.

If you encounter this exception, you may want to check the following:

Check your selector used in your find\_by...

Element may not yet be on the screen at the time of the find operation, (webpage is still loading) see selenium.webdriver.support.wait.WebDriverWait() for how to write a wait wrapper to wait for an element to appear.

exception selenium.common.exceptions.NoSuchFrameException(msg=None, screen=None, stacktrace=None)

Thrown when frame target to be switched doesn’t exist.

exception selenium.common.exceptions.NoSuchWindowException(msg=None, screen=None, stacktrace=None)

Thrown when window target to be switched doesn’t exist.

To find the current set of active window handles, you can get a list of the active window handles in the following way:

print driver.window\_handles

exception selenium.common.exceptions.RemoteDriverServerException(msg=None, screen=None, stacktrace=None)

exception selenium.common.exceptions.ScreenshotException(msg=None, screen=None, stacktrace=None)

A screen capture was made impossible.

exception selenium.common.exceptions.SessionNotCreatedException(msg=None, screen=None, stacktrace=None)

A new session could not be created.

exception selenium.common.exceptions.StaleElementReferenceException(msg=None, screen=None, stacktrace=None)

Thrown when a reference to an element is now “stale”.

Stale means the element no longer appears on the DOM of the page.

Possible causes of StaleElementReferenceException include, but not limited to:

You are no longer on the same page, or the page may have refreshed since the element was located.

The element may have been removed and re-added to the screen, since it was located. Such as an element being relocated. This can happen typically with a javascript framework when values are updated and the node is rebuilt.

Element may have been inside an iframe or another context which was refreshed.

exception selenium.common.exceptions.TimeoutException(msg=None, screen=None, stacktrace=None)

Thrown when a command does not complete in enough time.

exception selenium.common.exceptions.UnableToSetCookieException(msg=None, screen=None, stacktrace=None)

Thrown when a driver fails to set a cookie.

exception selenium.common.exceptions.UnexpectedAlertPresentException(msg=None, screen=None, stacktrace=None, alert\_text=None)

Thrown when an unexpected alert is appeared.

Usually raised when when an expected modal is blocking webdriver form executing any more commands.

exception selenium.common.exceptions.UnexpectedTagNameException(msg=None, screen=None, stacktrace=None)

Thrown when a support class did not get an expected web element.

exception selenium.common.exceptions.UnknownMethodException(msg=None, screen=None, stacktrace=None)

The requested command matched a known URL but did not match an method for that URL.

exception selenium.common.exceptions.WebDriverException(msg=None, screen=None, stacktrace=None)

Base webdriver exception.

##### **Writing your own exception class**

Now, let’s see how to create a custom exception in action. Here are the steps:

* Create a new class whose name should end with Exception like ClassNameException. This is a convention to differentiate an exception class from regular ones.
* Make the class extends one of the exceptions which are subtypes of the java.lang.Exception class. Generally, a custom exception class always extends directly from the Exception class.
* Create a constructor with a String parameter which is the detail message of the exception. In this constructor, simply call the super constructor and pass the message.

Let’s see a simple example of java custom exception.

class InvalidAgeException extends Exception{

InvalidAgeException(String s){

super(s);

}

}

class TestCustomException1{

static void validate(int age)throws InvalidAgeException{

if(age<18)

throw new InvalidAgeException("not valid");

else

System.out.println("welcome to vote");

}

public static void main(String args[]){

try{

validate(13);

}catch(Exception m){System.out.println("Exception occured: "+m);}

System.out.println("rest of the code...");

}

}

Output:

Exception occured: InvalidAgeException:not valid

rest of the code...