Top of Form

#### ****How to Handle an Exception:****

Java provides two different options to handle an exception. You can either use the try-catch-finally approach to handle all kinds of exceptions. Or you can use the try-with-resource approach which allows an easier cleanup process for resources.

##### **Try-Catch-Finally:**

That is the classical approach to handle an exception in Java. It can consist of 3 steps:

* a try block that encloses the code section which might throw an exception,
* one or more catch blocks that handle the exception and
* a finally block which gets executed after the try block was successfully executed or a thrown exception was handled.

The try block is required, and you can use it with or without a catch or finally block.

#### ****1.The Try Block****

Let’s talk about the try block first. It encloses the part of your code that might throw the exception. If your code throws more than one exception, you can choose if you want to:

* use a separate try block for each statement that could throw an exception or
* use one try block for multiple statements that might throw multiple exceptions.

The following example shows a try block which encloses three method calls.

**public void performBusinessOperation() {**

**try {**

**doSomething("A message");**

**doSomethingElse();**

**doEvenMore();**

**}**

**// see following examples for catch and finally blocks**

**}**

**public void doSomething(String input) throws MyBusinessException {**

**// do something useful ...**

**throw new MyBusinessException("A message that describes the error.");**

**}**

**public void doSomethingElse() {**

**// do something else ...**

**}**

**public void doEvenMore() throws NumberFormatException{**

**// do even more ...**

**}**

As you can see in the method definitions, only the first and the third method specify an exception. The first one might throw a MyBusinessException, and the doEvenMore method might throw a NumberFormatException.

In the next step, you can define one catch block for each exception class you want to handle and one finally block. All checked exceptions that are not handled by any of the catch blocks need to be specified.

#### ****2.The Catch Block****

You can implement the handling for one or more exception types within a catch block. As you can see in the following code snippet, the catch clause gets the exception as a parameter. You can reference it within the catch block by the parameter name.

**public void performBusinessOperation() {**

**try {**

**doSomething("A message");**

**doSomethingElse();**

**doEvenMore();**

**} catch (MyBusinessException e) {**

**e.printStackTrace();**

**} catch (NumberFormatException e) {**

**e.printStackTrace();**

**}**

**}**

The previous code sample shows two catch blocks. One to handle the MyBusinessException and one to handle the NumberFormatException. Both blocks handle the exceptions in the same way. Since Java 7, you can do the same with just one catch block.

**public void performBusinessOperation() {**

**try {**

**doSomething("A message");**

**doSomethingElse();**

**doEvenMore();**

**} catch (MyBusinessException|NumberFormatException e) {**

**e.printStackTrace();**

**}**

**}**

The implementation of the catch blocks in the previous examples is very basic. The printStackTrace method which writes the class, message and call stack of the exception to system out.

**com.stackify.example.MyBusinessException: A message that describes the error.**

**at com.stackify.example.TestExceptionHandling.doSomething(TestExceptionHandling.java:84)**

**at com.stackify.example.TestExceptionHandling.performBusinessOperation(TestExceptionHandling.java:25**)

In a real application, you might want to use a more advanced implementation. You can, for example, show an error message to the user and request a different input or you could write a record into the work log of your batch process. Sometimes, it might even be ok to catch and ignore the exception.

#### ****3.The Finally Block****

The finally block gets executed after the successful execution of the try block or after one of the catch blocks handled an exception. It is, therefore, a good place to implement any cleanup logic, like closing a connection or an InputStream.

You can see an example of such a cleanup operation in the following code snippet. The finally block will be executed, even if the instantiation of the FileInputStream throws a FileNotFoundException or the processing of the file content throws any other exception.

**FileInputStream inputStream = null;**

**try {**

**File file = new File("./tmp.txt");**

**inputStream = new FileInputStream(file);**

**// use the inputStream to read a file**

**} catch (FileNotFoundException e) {**

**e.printStackTrace();**

**} finally {**

**if (inputStream != null) {**

**try {**

**inputStream.close();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**}**

As you’ve seen, the finally block provides a good option to prevent any leaks. And before Java 7, it was a best practice to put all cleanup code into a finally block.

Bottom of Form

Top of Form

###### 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 see custom exceptions later.

The syntax of java throw keyword is given below.

**throw exception;**

Let's see the example of throw IOException.

**throw new IOException("sorry device error);**

**Println ("normal flow...")**

###### java throw keyword example:

In this example, we have created the validate method that takes an integer value as a parameter. If the age is less than 18, we are throwing the ArithmeticException otherwise print a message welcome to vote.

**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...");**

**}**

**}**

**throws Keyword (Exception Handling):**

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 has not performed check up before the code being used.

###### Syntax of java throws:

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

**//method code**

**}**

###### Advantage of Java throws keyword:

Now Checked Exception can be propagated (forwarded in call stack).

It provides information to the caller of the method about the exception.

###### 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...");**

**}**

**}**

Bottom of Form

Top of Form

Bottom of Form

Top of Form

##### **Reference Links:**

**ONLINE NOTES LINKS:**

<https://www.w3schools.com/java/java_try_catch.asp>

https://www.javatpoint.com/try-catch-block

https://docs.oracle.com/javase/tutorial/essential/exceptions/try.html

https://www.guru99.com/java-exception-handling.html

VIDEO LINKS:

<https://www.youtube.com/watch?v=4my7mKFaNQs>

<https://www.youtube.com/watch?v=W-N2ltgU-X4>

Bottom of Form

Help for current page