**Exception handling in java-**

Exception is the abnormal condition that occur during execution of program to stop the entire flow of application called as “Exception.”

The Exception Handling in Java is one of the powerful mechanism to handle the runtime errors so that the normal flow of the application can be maintained.

It is highly recommended to handle exceptions. The main objective of exception handling is normal (graceful) termination of the program.

Exception handling doesn't mean repairing an exception. We have to define alternative way to continue rest of the program normally. This way of defining alternative is nothing but exception handling

Why?

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

System.***out***.println("First line");

System.***out***.println("Second line");

System.***out***.println("Third line");

}

}

Output-

First line

Second line

Third line

Now we add one exception line code as below

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

System.***out***.println("First line");

System.***out***.println("Second line");

System.***out***.println("Third line");

**int** a = 10 / 0;

System.***out***.println("Fourth line");

System.***out***.println("Fifth line");

}

}

Output-

First line

Second line

Third line

Exception in thread "main" java.lang.ArithmeticException: / by zero at com.test.Test.main(Test.java:10)

In this example, two statements are not executed, if you want to execute that two statements then how to do it in java?

Then you should go for exception handling.

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

System.***out***.println("First line");

System.***out***.println("Second line");

System.***out***.println("Third line");

**try** {

**int** a = 10 / 0;

} **catch** (Exception e) {

System.***out***.println(e);

}

System.***out***.println("Fourth line");

System.***out***.println("Fifth line");

}

}

Output-

First line

Second line

Third line

java.lang.ArithmeticException: / by zero

Fourth line

Fifth line

Here two statements are executed i.e fourth line and fifth line, we have achieved this by using try and catch block.

**How Does JVM handle an Exception?**

Default Exception Handling:

Whenever inside a method, if an exception has occurred, the method creates an Object known as an Exception Object and hands it to JVM.

The exception object contains following information:

1. the name and description of the exception
2. the location of exeption.

Creating the Exception Object and handling it in the JVM is called throwing an Exception.

The run-time system searches the call stack to find the method that contains a block of code that can handle the occurred exception. The block of the code is called an Exception handler.

The JVM starts searching from the method in which the exception occurred, and proceeds through the call stack in the reverse order in which methods were called.

If it finds an appropriate handler, then it passes the occurred exception to it. An appropriate handler means the type of the exception object thrown matches the type of the exception object it can handle.

If the run-time system searches all the methods on the call stack and couldn’t have found the appropriate handler, then the run-time system handover the Exception Object to the default exception handler, which is part of the run-time system.

This handler prints the exception information in the following format and terminates the program abnormally.

**Exception hierarchy-**



**Throwable-**

* In the above given Hierarchy Throwable is a class which is at the top of the exception hierarchy, from which all exception classes are derived.
* It is the super class of all Exceptions in Java.
* Both Exception and Errors are java classes which are derived from the Throwable class.

**Error-**

* Error is subclass of throwable class.
* Errors are mostly the abnormal conditions.
* Error does not occur because of the programmer’s mistakes, but when system is not working properly or a resource is not allocated properly.
* Memory out of bound exception, stack overflow etc., are examples of Error.

**Difference between Checked and Unchecked Exceptions**

**1) Checked Exception**

The classes which directly inherit Throwable class except RuntimeException and Error are known as checked exceptions

The exceptions which are checked by the compiler whether programmer handling or not, for smooth execution of the program at runtime, are called checked exceptions.

Example-

IOException

SQLException

**2) Unchecked Exception**

The classes which inherit RuntimeException are known as unchecked exceptions.

Unchecked exceptions are not checked by the compiler whether programmer handing or not, but they are checked at runtime.

Example-

ArithmeticException-

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int** a = 10 / 0;

}

}

Output

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

at com.test.Test.main(Test.java:7)

NullPointerException

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

String str = **null**;

System.***out***.println(str.length()); //exception will be occured.

}

}

Output

Exception in thread "main" java.lang.NullPointerException

at com.test.Test.main(Test.java:8)

ArrayIndexOutOfBoundsException

**package** com.test;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

**int** a[] = **new** **int**[5];

a[10] = 50; // ArrayIndexOutOfBoundsException

}

}

output

Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 10

at com.test.Test.main(Test.java:8)

3) Error

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

**Possible way to write try catch block**

1.

try {

//not allowed

}

2.

try {

//allowed

}

catch(Exception e){

}

3.

try {

//allowed

}

finally{

}

4.

try{

//allowed

}

catch (Exception e){

}

finally{

}

5.

try {

//allowed

}

Catch(ArithmaticException e1){

}

Catch(Exception e){

}

6.

try {

//not allowed

}

Catch(Exception e){

}

Catch(ArithmaticException e1){

}

Not allowed – Reason: The bigger exception cannot be in the first catch because it will accommodate all exceptions and there will be no chance to reach the second catch of NullpointerException

try{

}

catch(Exception e) {

}

catch(NullPointerException npe) {

}

8.

try {

// allowed

}

Catch(ArithmaticException e){

}

Catch(Exception e1){

}

**finally-**

The finally block is used when an important part of the code needs to be executed. It is always executed whether or not the exceptions are handled.

* Finally block will always get executed until we shut down JVM. To shut down JVM in java we call System.exit (). If you write this in try block in that case finally block will not be executed.
* Normally, finally block contains the code to release resources like DB connections, IO streams etc

Question. What is the difference Between Catch and finally in java?

|  |  |  |
| --- | --- | --- |
| Sr. No. | Catch | Finally |
| 1 | Catch block handles the error when it occurs in try block | There is no need of exception thrown by try block |
| 2 | Catch block is executed only when the if exception is thrown by try block, otherwise it is not executed | Finally block is always executed whether exception occurs or not |
| 3 | We can use multiple catch block for only one try block | Only one finally block is used for one try block |
| 4 | We can handle multiple exceptions by using catch blocks | It is not for exception handling |

Q. 1

int m1() {

try {

return 10;

} catch (Exception e) {

return 20;

} finally {

return 30;

}

}

return-30

Q 2

public class FinallyTest {

int m1() {

try {

return 10;

} catch (Exception e) {

return 20;

} finally {

return 30;

}

return 40;

}

}

It will not compile, unreachable code return 40.