Exception Handling

## ***Exception***:

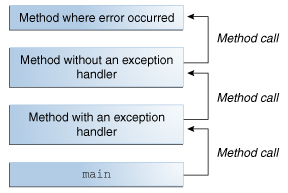
The term exception is shorthand for the phrase "exceptional event" which occurs at runtime.

**Definition**: An exception is an un wanted or un expected event, which occurs during the execution of a program and disturbs the normal flow of the program.

When an exception occurs, JVM creates an exception object which contains the information like type , name of the exception, description and the method in which it occurred.

## Default Exception Handler Mechanism:

JVM creates an exception object and hand overs it to runtime system. Then runtime system checks in the corresponding method, whether it contains exception handling mechanism or not. If it doesn’t contain any exception handling mechanism then it throws the exception object to its calling method. The list of method calls is known as **Call Stack**.



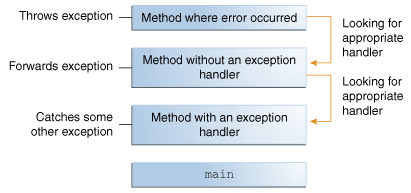
Call stack Image

**Exception handler** is nothing but a block of code i.e catch block, which handles the exception or specifies an alternate program flow.

The runtime system searches the call stack exception handler. The search begins with the method in which the error occurred and proceeds through the call stack in the reverse order in which the methods were called. When an appropriate handler is found, the runtime system passes the exception to the handler.

An exception handler is considered appropriate if the type of the exception object thrown matches the type that can be handled by the handler.

If the runtime system is unable to find the exception handler in the call stack, then the program terminates abnormally from main() method.



Searching the call stack for the exception handler.

## TYPES OF EXCEPTIONS

1. Checked Exception
2. Runtime Exception
3. Error

### Checked exception:

Checked exceptions are exceptions which are anticipated during the compile time itself.

Ex: FileNotFoundException .. we may get this error, incase the required file is not available.

All exceptions are checked exceptions, except for those indicated by Error, RuntimeException, and their subclasses.

These exceptions must be handled by using try- catch or by using throws clause otherwise will get compile time error.

### **Unchecked exceptions:**

Errors and runtime exceptions are collectively known as unchecked exceptions.

**Runtime Exceptions:**

These exceptions are internal to the application.

These usually indicate programming bugs, such as logic errors or improper use of an API.

Ex : NullPointerException.

**Error:**

These are exceptional conditions that are external to the application, and that the application usually cannot anticipate or recover from because of a hardware or system malfunction.

Ex: OutofMemoryError.

**Exception Hierarchy Image**

## Exception Handling Key words:

Try, catch , finally, throw, throws.

### Try key word:

Here, try block will contain the risky codethat might throw an exception within a try block.

try {

statement(s) which throws an exception

}

catch and finally blocks . . .

In try, each line of code will be executed until an exception occurred.

ExceptionHandlingSample1.java

For example, as shown in the above program

While using ' / ' operator, there may be a chance to get an Arithmetic exception. So we placed that division operation logic with in the try block.

### Catch key word:

Every try block will be followed by one or more catch blocks directly.

We should not any write any statements/code between the end of the try block and the beginning of the catch block.

try {

} catch (ExceptionType name) {

} catch (ExceptionType name) {

}

Catch block’s exception type will specify the exception class name for which it invokes the catch blcok.

Execution of the catch block code is optional.

The runtime system executes the specific catch block, if and only if the exceptionType matches the type of the exception thrown.

Within the catch block, we can write the exception info to the loggers/console.

In case of multiple catch blocks, the order of the catch blocks will also places a major role.

First we need to handle the child class exception types, next we need to handle the parent class exception types.

ExceptionHandlingSample1.java

For example, as shown in the above program

Un comment the exception catch block, then will get Compile time error.Because Exception class is the parent class for ArithmeticException.

### Finally key word:

The finally block will always executes along with the try block, If we get exception or not.

It’s always a good practice to close the resources within the finally block.

**system.exit(0); // this statement will not allow to execute the finally block.**

ExceptionHandlingSample1.java

For example, as shown in the above program

If we run the above program directly, it will executes the finally block without any issues.

Next, un comment the line ... system.exit(0) and run the program again

Now this time it won’t execute the finally block.

NOTE: The finally block is a key tool for preventing resource leaks.

ExceptionHandlingSample1.java

Explanation: If you run the above program, will get the Arithmetic exception at below piece of code.

result = 62 / divisor;

Then the remaining statements in the try block will not be executed.

AS we already specified the catch block with Arithmetic Exception type, above exception will be caught here. Will take necessary actions in the corresponding catch block based on requirement.

Once it completes the catch block, then it will execute the finally block.

**Output screen shot**

Scenario – 1: Uncomment the catch block which is of type Exception.

After uncommenting the catch block, then we will get Compile time error at the second catch block as

**CTE : Un reachable catch block for Arithmetic Exception.**

While defining multiple catch blocks we must follow the order of exception class hierarchy.

To resolve this error, Place the arithmetic exception catch block before the catch block of exception type as below.

**catch** (ArithmeticException e) {

System.***out***.println("We got Arithmetic Exception");

}

**catch**(Exception e){

System.***out***.println("Invalid order of the catch blocks");

}

Scenario – 2: Uncomment the system.exit(0) code base and run the program again.

As we already know, this will not execute the finally block code.

**Output screen shot**

### Throw key word:

We will use the **throw** keyword to throw an exception from any method.

Syntax : throw *someThrowableObject*;

CustomExceptionSample.java

In validateJavaVersion() method, we are checking the version of java and using the **throw** keyword to throw the InvalidJavaVersion Exception (It’s a custom exception)

### Throws key word:

We will use throws keyword to navigate the exception handling mechanism to caller method.

Will use throws keyword for checked exceptions only.

CustomExceptionSample.java

For example, validateJavaVersion() method throws an InvalidJavaVersion Exception to the caller method.