**Exception handling in java with example programs**

BY CHAITANYA SINGH | FILED UNDER: [**EXCEPTION HANDLING**](http://beginnersbook.com/category/technology/java-guide/exception-handling/)

In this tutorial we will discuss what is an exception and how it can be handled in java programming language.

**What is an exception?**

An Exception can be anything which interrupts the normal flow of the program. When an exception occurs program processing gets terminated and doesn’t continue further. In such cases we get a system generated error message. The good thing about exceptions is that they can be handled. We will cover the handling part later in this same tutorial.

**When an exception can occur?**  
Exception can occur at runtime (known as runtime exceptions) as well as at compile-time (known Compile-time exceptions).

**Reasons for Exceptions**  
There can be several reasons for an exception. For example, following situations can cause an exception – Opening a non-existing file, Network connection problem, Operands being manipulated are out of prescribed ranges, class file missing which was supposed to be loaded and so on.

**Difference between error and exception**

**Errors** indicate serious problems and abnormal conditions that most applications should not try to handle. Error defines problems that are not expected to be caught under normal circumstances by our program. For example memory error, hardware error, JVM error etc.  
**Exceptions** are conditions within the code. A developer can handle such conditions and take necessary corrective actions. Few examples –

* DivideByZero exception
* NullPointerException
* ArithmeticException
* ArrayIndexOutOfBoundsException

**Advantages of Exception Handling**

* Exception handling allows us to control the normal flow of the program by using exception handling in program.
* It throws an exception whenever a calling method encounters an error providing that the calling method takes care of that error.
* It also gives us the scope of organizing and differentiating between different error types using a separate block of codes. This is done with the help of try-catch blocks.

**Why to handle exception?**  
If an exception is raised, which has not been handled by programmer then program execution can get terminated and system prints a non user friendly error message.

Ex:-Take a look at the below system generated exception

**An exception generated by the system is given below**

Exception in thread "main" java.lang.ArithmeticException: / by zero at ExceptionDemo.main(ExceptionDemo.java:5)

ExceptionDemo : The class name

main : The method name

ExceptionDemo.java : The filename

java:5 : Line number

For a novice user the above message won’t be easy to understand. In order to let them know that what went wrong we use exception handling in java program. We handle such conditions and then prints a user friendly warning message to user, which lets them correct the error as most of the time exception occurs due to bad data provided by user.

**Types of exceptions**

There are two types of exceptions

1)Checked exceptions  
2)Unchecked exceptions

Below is a brief about each however if you want a detailed tutorial with examples then you can refer [**Checked and Unchecked exceptions in Java**](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/).

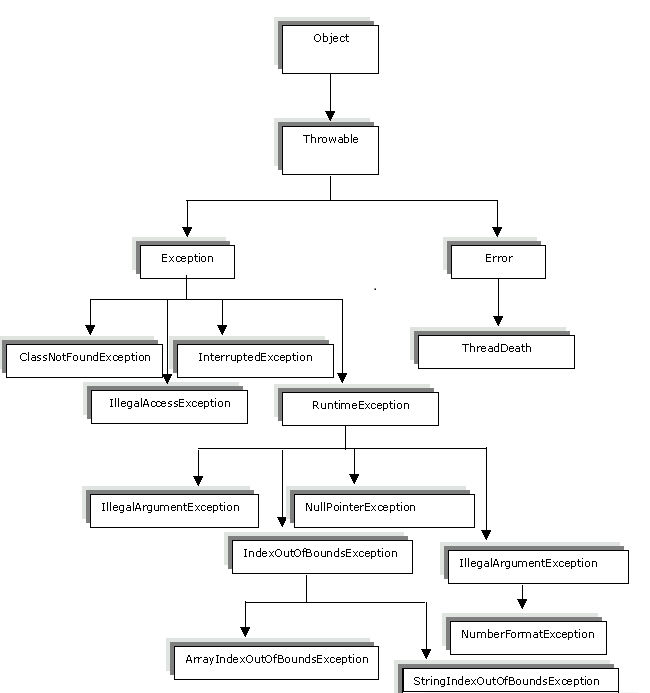
**Checked exceptions**  
All exceptions other than Runtime Exceptions are known as Checked exceptions as the compiler checks them during compilation to see whether the programmer has handled them or not. If these exceptions are not handled/declared in the program, it will give compilation error.

**Examples of Checked Exceptions :-**  
ClassNotFoundException  
IllegalAccessException  
NoSuchFieldException  
EOFException etc.

**Unchecked Exceptions**  
Runtime Exceptions are also known as Unchecked Exceptions as the compiler do not check whether the programmer has handled them or not but it’s the duty of the programmer to handle these exceptions and provide a safe exit.  
These exceptions need not be included in any method’s throws list because compiler does not check to see if a method handles or throws these exceptions.

**Examples of Unchecked Exceptions:-**  
ArithmeticException  
ArrayIndexOutOfBoundsException  
NullPointerException  
NegativeArraySizeException etc.

**Exception hierarchy**



**Exception handling in Java**

Here are the list of tutorials which will help you understand the exception handling part in detail. Read them in the given sequence to grasp the concepts easily.

1. [**Try-catch in Java**](http://beginnersbook.com/2013/04/try-catch-in-java/)
2. [**Nested Try Catch**](http://beginnersbook.com/2013/04/nested-try-catch/)
3. [**Checked and unchecked exceptions**](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/)
4. [**Finally block in Java**](http://beginnersbook.com/2013/04/java-finally-block/)
5. [**try-catch-finally**](http://beginnersbook.com/2013/05/flow-in-try-catch-finally/)
6. [**finally block & return statement**](http://beginnersbook.com/2013/05/java-finally-return/)
7. [**Throw exception in Java**](http://beginnersbook.com/2013/04/throw-in-java/)
8. [**Example of throw keyword**](http://beginnersbook.com/2013/12/throw-keyword-example-in-java/)
9. [**Example of throws clause**](http://beginnersbook.com/2013/12/throws-keyword-example-in-java/)
10. [**Throws in Java**](http://beginnersbook.com/2013/04/java-throws/)
11. [**throw vs throws**](http://beginnersbook.com/2013/04/difference-between-throw-and-throws-in-java/)
12. [**Exception handling examples**](http://beginnersbook.com/2013/04/exception-handling-examples/)

**Try Catch in Java – Exception handling**

BY CHAITANYA SINGH | FILED UNDER: [**EXCEPTION HANDLING**](http://beginnersbook.com/category/technology/java-guide/exception-handling/)

**What is Try Block?**

The try block contains a block of program statements within which an exception might occur. A try block is always followed by a catch block, which handles the exception that occurs in associated try block. A try block must followed by a Catch block or Finally block or both.

**Syntax of try block**

try{

//statements that may cause an exception

}

**What is Catch Block?**

A catch block must be associated with a try block. The corresponding catch block executes if an exception of a particular type occurs within the try block. For example if an [**arithmetic exception**](http://beginnersbook.com/2013/04/exception-handling-examples/) occurs in try block then the statements enclosed in catch block for arithmetic exception executes.

**Syntax of try catch in java**

try

{

//statements that may cause an exception

}

catch (exception(type) e(object))‏

{

//error handling code

}

**Flow of try catch block**

1. If an exception occurs in try block then the control of execution is passed to the catch block from try block. The exception is caught up by the corresponding catch block. A single try block can have multiple catch statements associated with it, but each catch block can be defined for only one exception class. The program can also contain [**nested**](http://beginnersbook.com/2013/04/nested-try-catch/) [**try-catch-finally blocks**](http://beginnersbook.com/2013/05/flow-in-try-catch-finally/).
2. After the execution of all the try blocks, the code inside the finally block executes. It is not mandatory to include a finally [**block**](http://beginnersbook.com/2013/04/java-finally-block/) at all, but if you do, it will run regardless of whether an exception was thrown and handled by the try and catch blocks.

**An example of Try catch in Java**

class Example1 {

public static void main(String args[]) {

int num1, num2;

try {

// Try block to handle code that may cause exception

num1 = 0;

num2 = 62 / num1;

System.out.println("Try block message");

} catch (ArithmeticException e) {

// This block is to catch divide-by-zero error

System.out.println("Error: Don't divide a number by zero");

}

System.out.println("I'm out of try-catch block in Java.");

}

}

Output:

Error: Don't divide a number by zero

I'm out of try-catch block in Java.

**Multiple catch blocks in Java**

1. A try block can have any number of catch blocks.  
2. A catch block that is written for catching the class Exception can catch all other exceptions  
Syntax:

catch(Exception e){

  //This catch block catches all the exceptions

}

3. If multiple catch blocks are present in a program then the above mentioned catch block should be placed at the last as per the exception handling best practices.  
4. If the try block is not [**throwing any exception**](http://beginnersbook.com/2013/04/throw-in-java/), the catch block will be completely ignored and the program continues.  
5. If the try block [**throws an exception**](http://beginnersbook.com/2013/04/java-throws/), the appropriate catch block (if one exists) will catch it  
–catch(ArithmeticException e) is a catch block that can catch ArithmeticException  
–catch(NullPointerException e) is a catch block that can catch NullPointerException  
6. All the statements in the catch block will be executed and then the program continues.

**Example of Multiple catch blocks**

class Example2{

public static void main(String args[]){

try{

int a[]=new int[7];

a[4]=30/0;

System.out.println("First print statement in try block");

}

catch(ArithmeticException e){

System.out.println("Warning: ArithmeticException");

}

catch(ArrayIndexOutOfBoundsException e){

System.out.println("Warning: ArrayIndexOutOfBoundsException");

}

catch(Exception e){

System.out.println("Warning: Some Other exception");

}

System.out.println("Out of try-catch block...");

}

}

Output:

Warning: ArithmeticException

Out of try-catch block...

In the above example there are multiple catch blocks and these catch blocks executes sequentially when an exception occurs in try block. Which means if you put the last catch block ( catch(Exception e)) at the first place, just after try block then in case of any exception this block will execute as it has the [**ability to handle all exceptions**](http://beginnersbook.com/2013/04/java-exception-handling/). This catch block should be placed at the last to avoid such situations.