**Exception handling in java**

## Introduction

Errors are the wrongs that can make a program to go wrong.An error may produce an incorrect output or may terminate the execution of the program abruptly or even may cause the system to crash.

**Types of Errors**

Errors may broadly be classified into two categories..

* Compile-time errors
* Run-time errors

**Compile-Time Errors**

All syntax errors will be detected and displayed by the Java compiler and therefore these errors are known as compile-time errors. Whenever the compiler displays an error, it will not create the .class file. It is therefore necessary that we fix all the errors before we can successfully compile and run the program..

* Most of the compile-time errors are due to typing mistakes. Typographical errors are hard to find. We may have to check the code word by word, of even character by character .The most common problems are:
* Missing semicolons
* Missing (or mismatch of ) brackets in classes and methods
* Misspelling of identifiers and keywords
* Missing double quotes in strings
* Use of undeclared variables
* Incompatible types in assignments / initialization
* Bad references to objects
* Use of = in place of == operator
* And so on

**Run-Time Errors**

Sometimes a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stackOverflow. Most common run-time errors are:

* Dividing an integer by zero
* Accessing an element that is out of the bounds of an array
* Trying to store a an value into an array of incompatible class or type
* Trying to cast an instance of a class to one of its subclasses
* Passing a parameter that is not in a valid range or value for a method
* Trying to illegally change the state of a thread
* Attempting to use a negative size for an array.
* Using a null object reference as a legitimate object reference to access a method or a variable.
* Converting invalid string to a number
* Accessing a character that is out of bounds of a string
* And may more

When such errors are encountered, Java typically generates an error message and aborts the program.

## Exception

An exception is a condition that is caused by a run-time error in the program. When the java interpreter encounters an runtime-error, it creates an exception object throws it.

If the exception object is not caught and handled properly, the interpreter will display an error message and will terminate the program. if we want the program to continue with the execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then display an appropriate message for taking corrective actions. This task is known as exception handling.

The purpose of exception handling mechanism is to provide a means to detect and report an “exceptional circumstance” so that appropriate action can be taken. The mechanism suggests incorporation of a separate error handling code that performs the following tasks:

1. Find the problem (Hit the exception).
2. Inform that an error has occurred (Throw the exception)
3. Receive the error information (Catch the exception)
4. Take corrective actions (Handle the exception)

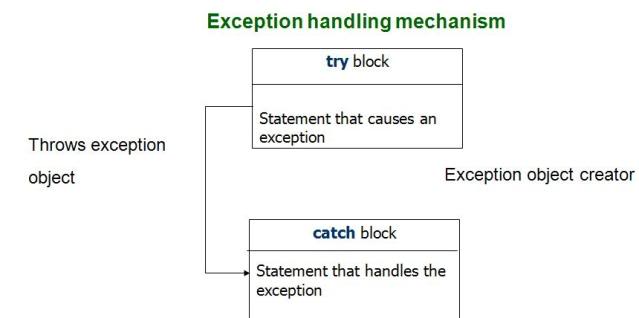
The error handling code basically consists of two segments, one to detect errors and to throw exceptions and the other to catch exceptions and to take appropriate actions.

**Some of Common Java Exceptions**

|  |  |
| --- | --- |
| [ArithmeticException](http://rymden.nu/exceptions.html#ArithmeticException) | Caused by Math Errors such as Division by zero |
| [ArrayIndexOutOfBoundsException](http://rymden.nu/exceptions.html#ArrayIndexOutOfBoundsException) | Caused by bad array indexes. |
| [ArrayStoreException](http://rymden.nu/exceptions.html#ArrayStoreException) | Caused when a program tries to store the wrong type of data in an array |
| [ClassNotFoundException](http://rymden.nu/exceptions.html#ClassNotFoundException) | Caused when the required class is not found. |
| [NoSuchMethodException](http://rymden.nu/exceptions.html#NoSuchMethodException) | Caused when the method called is not found |
| [NullPointerException](http://rymden.nu/exceptions.html#NullPointerException) | Caused by referencing a null object |
| [NumberFormatException](http://rymden.nu/exceptions.html#NumberFormatException) | Caused when a conversion between strings and numbers fails |
| IOException | Caused by general I/O failures |
| FileNotFoundException | Caused by an attempting to access a non existing file |

**Syntax of Exception handling Code**

The basic concepts of exception handling are throwing an exception and catching it.



**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 :

try{

//statements that may cause an exception

}

### 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 {

num1 = 0;

num2 = 62 / num1;

System.out.println("Result="+num2);

}

catch (ArithmeticException e) {

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

}

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

}

}

**Multiple Catch Statements**

It is possible to have more than one catch statement in the catch block as explained below

.............

..............

try

{

Statement ; // generates an exception

}

catch( Exception-Type-1 e)

{

Statement ; // processes exception Type-1

}

catch( Exception-Type-2 e)

{

Statement ; // processes exception Type-2

}

catch( Exception-Type-n e)

{

Statement ; // processes exception Type-n

}

When an exception in a try block is generated, the java treats the multiple catch statements like cases in a switch statement. The first statement whose parameter matches with the exception object will be executetd and the remaining statements will be skipped.

**Example**

**class ExceptionExample**

**{**

**public static void main(String argv[])**

**{**

**int num1 = 10;**

**int num2 = 0;**

**int result = 0;**

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

**try {**

**arr[0] = 0;**

**arr[1] = 1;**

**arr[2] = 2;**

**arr[3] = 3;**

**arr[4] = 4;**

**arr[5] = 5;**

**result = num1 / num2;**

**throw MyException;**

**System.out.println("Result of Division : " + result);**

**}**

**catch (ArithmeticException e)**

**{**

**System.out.println("Err: Divided by Zero");**

**}**

**catch (ArrayIndexOutOfBoundsException e)**

**{**

**System.out.println("Err: Array Out of Bound");**

**}**

**Catch(MyException m)**

**{**

**}**

**}**

## Using Finally statement

1. A [finally statement](http://beginnersbook.com/2013/05/java-finally-return/) must be associated with a [try statement](http://beginnersbook.com/2013/04/try-catch-in-java/). It identifies a block of statements that needs to be executed regardless of whether or not an [exception occurs](http://beginnersbook.com/2013/04/java-exception-handling/) within the try block.

2. After all other try-catch processing is complete, the [code inside the finally block executes](http://beginnersbook.com/2013/05/flow-in-try-catch-finally/). It is not mandatory to include a 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 parts of the block.

3. In normal execution the finally block is executed after try block. When any exception occurs first the catch block is executed and then finally block is executed.

4. An exception in the finally block, exactly [behaves like any other exception](http://beginnersbook.com/2013/04/user-defined-exception-in-java/).

5. The code present in the **finally block** executes even if the try or catch block contains control transfer statements like [return](http://beginnersbook.com/2013/05/java-finally-return/), break or continue.

To understand above concepts better refer the below [examples](http://beginnersbook.com/2013/04/exception-handling-examples/).

#### Syntax:

try

{

//statements that may cause an exception

}

finally

{

//statements to be executed

}

# Throwing our own exceptions

**There may be times when we would like to throw our own exceptions.** These exceptions are known as [user-defined exceptions](http://beginnersbook.com/2013/04/user-defined-exception-in-java/). In order to throw user defined exceptions, [throw keyword](http://beginnersbook.com/2013/12/throw-keyword-example-in-java/) is being used. we will see how to create a new exception and throw it in a program using **throw keyword.**

You can also throw an already defined exception like ArithmeticException, IOException etc.

## Syntax

throw new throwable\_subclass;

**Examples:**

throw ( new ArithmeticException);

throw new NumberFormatException( );

**Flow of execution while throwing an exception using throw keyword**

Whenever a throw statement is encountered in a program the next statement doesn’t execute. Control immediately transferred to catch block to see if the thrown exception is handled there. If the exception is not handled there then next catch block is being checked for exception and so on. If none of the [catch block](http://beginnersbook.com/2013/04/try-catch-in-java/) is handling the thrown exception then a system generated exception message is being populated on screen, same what we get for un-handled exceptions.  
 Example:

class MyException extends Exception

{

MyException(String message)

{

super(message);

}

}

class TestMyException

{

public static void main(String args[])

{

int x=2,y=1000;

try

{

float z=(float) x/(float) y;

if(z<0.01 )

throw new MyException("Number is too small..." );

}

catch( MyException e)

{

System.out.println("Caught my Exception..");

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

}

}

}