**How to Catch multiple exceptions**

[**EXCEPTION HANDLING**](http://beginnersbook.com/category/technology/java-guide/exception-handling/)

A method can [**throw more than one exceptions**](http://beginnersbook.com/2013/04/throw-in-java/). However that method needs to declare all the [**checked exceptions**](http://beginnersbook.com/2013/04/java-checked-unchecked-exceptions-with-examples/) it can throw (optionally method can declare the super class of exception if it is common among all the exceptions).

**Catching multiple exceptions**

lets have a look at the below example to understand it better:

**Example1**

public class MyExceptionClass

{

public void myMethod() throws ArithmeticException, ArrayIndexOutOfBoundsException

{

.....

.....

}

}

public class Demo

{

public void display()

{

MyExceptionClass obj= new MyExceptionClass();

try

{

//calling method of parent class

obj.myMethod();

}catch(ArithmeticException ex1)

{

System.out.println("Arithmetic Exception occurred!!");

}

catch(ArrayIndexOutOfBoundsException ex2)

{

System.out.println("Array Index Out of Bounds exception occurred!!");

}

}

}

**Exceptions are polymorphic in nature**

Now I’m going to rewrite the above example in different form:  
**>Example2**

public class MyExceptionClass

{

public void myMethod() throws ArithmeticException, ArrayIndexOutOfBoundsException

{

.....

.....

}

}

public class Demo

{

public void display()

{

MyExceptionClass obj= new MyExceptionClass();

try

{

//calling method of parent class

obj.myMethod();

}catch(Exception ex)

{

System.out.println("An Exception occurred");

}

}

}

By using only one catch block I have [**handled both the exceptions**](http://beginnersbook.com/2013/04/java-exception-handling/) but ask yourself is it good? off course no! As you won’t be able to provide recovery code(or error message) to the user as per specific exception. So if you compare above two examples you will find the example 1 is following the best practices of exception handling.

**Now we will see how to form a good sequence of catch blocks while exception handling**

public class Sample1

{

public void myMethod() throws ArithmeticException,

NullPointerException, ArrayIndexOutOfBoundsException

{

//Code

}

}

public class Sample2

{

public void myMethod2()

{

Sample1 obj= new Sample1();

try

{

//try block

obj.myMethod();

}catch(ArithmeticException ex1)

{

System.out.println("Arithmetic Exception occurred");

}

catch(NullPointerException ex2){

System.out.println("NullPointer exception occurred");

}

catch(ArrayIndexOutOfBoundsException ex2){

System.out.println("Array Index Out of Bounds exception occurred");

}

catch(Exception ex){

System.out.println("An exception occurred");

}

}

}

Whenever an [**exception**](http://beginnersbook.com/2013/04/exception-handling-examples/) occurs in [**try block**](http://beginnersbook.com/2013/04/try-catch-in-java/) then compiler jumps to the first catch block and from there it sequentially checks the catch block one by one, if it finds a catch block which can handle the raised exception then it executes the code inside that catch block. If Compiler doesn’t find any suitable catch block then it shows a system generated message.

**Note:** If you notice the sequence of catch blocks in above example then you will find that the super class( Exception class) [**catch block**](http://beginnersbook.com/2013/04/nested-try-catch/)is at last. Why is it so?? It’s because this catch block can handle all the [**exceptions which may occur**](http://beginnersbook.com/2013/04/user-defined-exception-in-java/) in this class so if you put it at first place it will execute every time an exception occurs and the specific catch blocks will never get a chance to execute. Hence the best practice is to put the default handler at the last place.