**Nested try catch: Java exception handling**

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

The [**try catch blocks can be nested**](http://beginnersbook.com/2013/04/try-catch-in-java/). One try-catch block can be present in the another try’s body. This is called **Nesting of try catch** blocks. Each time a try block does not have a catch handler for a [**particular exception**](http://beginnersbook.com/2013/04/user-defined-exception-in-java/), the stack is unwound and the next try block’s catch (i.e., parent try block’s catch) handlers are inspected for a match.

If no catch block matches, then the [**java run-time system**](http://beginnersbook.com/2013/04/runtime-compile-time-polymorphism/) will [**handle the exception**](http://beginnersbook.com/2013/04/java-exception-handling/). Lets see the syntax first then we will discuss this with an example.

**Syntax of Nested try Catch**

....

//Main try block

try

{

statement 1;

statement 2;

//try-catch block inside another try block

try

{

statement 3;

statement 4;

}

catch(Exception e1)

{

//Exception Message

}

//try-catch block inside another try block

try

{

statement 5;

statement 6;

}

catch(Exception e2)

{

//Exception Message

}

}

catch(Exception e3) //Catch of Main(parent) try block

{

//Exception Message

}

....

**Nested try catch example – explanation**

class Nest{

public static void main(String args[]){

//Parent try block

try{

//Child try block1

try{

System.out.println("Inside block1");

int b =45/0;

System.out.println(b);

}

catch(ArithmeticException e1){

System.out.println("Exception: e1");

}

//Child try block2

try{

System.out.println("Inside block2");

int b =45/0;

System.out.println(b);

}

catch(ArrayIndexOutOfBoundsException e2){

System.out.println("Exception: e2");

}

System.out.println("Just other statement");

}

catch(ArithmeticException e3){

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

System.out.println("Inside parent try catch block");

}

catch(ArrayIndexOutOfBoundsException e4){

System.out.println("ArrayIndexOutOfBoundsException");

System.out.println("Inside parent try catch block");

}

catch(Exception e5){

System.out.println("Exception");

System.out.println("Inside parent try catch block");

}

System.out.println("Next statement..");

}

}

**Output:**

Inside block1

Exception: e1

Inside block2

Arithmetic Exception

Inside parent try catch block

Next statement..

The above example shows Nested try catch use in Java. You can see that there are two try-catch block inside main try block’s body. I’ve marked them as block 1 and block 2 in above example.  
**Block1:**I have divided an integer by zero and it caused an arithmetic exception however the catch of block1 is handling arithmetic exception so"Exception: e1" got printed.

**Block2:**In block2 also, ArithmeticException occurred but block 2 catch is only handling ArrayIndexOutOfBoundsException so in this case control jump back to Main try-catch(parent) body. Since catch of parent try block is handling this exception that’s why “Inside parent try catch block” got printed as output.

**Parent try Catch block:**Since all the exception handled properly so program control didn’t get terminated at any point and at last “Next statement..” came as output.

**Note:** The main point to note here is that whenever the child try-catch blocks are not handling any exception, the control comes back to the parent try-catch if the exception is not handled there also then the program will terminate abruptly.

**Consider this example:**  
Here we have deep (two level) nesting which means we have a try-catch block inside a child try block. To make you understand better I have given the names to each try block in comments like try-block2 etc.

This is how the structure is: try-block3 is inside try-block2 and try-block2 is inside main try-block, you can say that the main try-block is a grand parent of the try-block3. Refer the explanation which is given at the end of this code.

class NestingDemo{

public static void main(String args[]){

//main try-block

try{

//try-block2

try{

//try-block3

try{

int arr[]= {1,2,3,4};

/\* I'm trying to display the value of

\* an element which doesn't exist. The

\* code should throw an exception

\*/

System.out.println(arr[10]);

}catch(ArithmeticException e){

System.out.print("Arithmetic Exception");

System.out.println(" handled in try-block3");

}

}

catch(ArithmeticException e){

System.out.print("Arithmetic Exception");

System.out.println(" handled in try-block2");

}

}

catch(ArithmeticException e3){

System.out.print("Arithmetic Exception");

System.out.println(" handled in main try-block");

}

catch(ArrayIndexOutOfBoundsException e4){

System.out.print("ArrayIndexOutOfBoundsException");

System.out.println(" handled in main try-block");

}

catch(Exception e5){

System.out.print("Exception");

System.out.println(" handled in main try-block");

}

}

}

Output:

ArrayIndexOutOfBoundsException handled in main try-block

As you can see that the ArrayIndexOutOfBoundsException has occurred in the grand child try-block3. Since try-block3 is not handling this exception, the control then gets transferred to the parent try-block2 and looked for the catch handlers in try-block2. Since the try-block2 is also not handling that exception, the control got transferred to the main grand parent try-block where it found the appropriate catch block for exception. This is how the**routing of exception is done in nested structure**.