Nested try catch block in Java – Exception handling

BY CHAITANYA SINGH | FILED UNDER: EXCEPTION HANDLING

When a try catch block is present in another try block then it is called the nested try catch block. Each time a try block does not have a catch handler for a particular exception, then the catch blocks of parent try block are inspected for that exception, if match is found that that catch block executes.

If neither catch block nor parent catch block handles exception then the system generated message would be shown for the exception, similar to what we see when we don't handle exception.

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;
      //try-catch block inside nested try block
      try {
         statement 5;
         statement 6;
      }
      catch(Exception e2) {
         //Exception Message
      }
   }
   catch(Exception e1) {
       //Exception Message
   }

}
//Catch of Main(parent) try block
catch(Exception e3) {
      //Exception Message
}
....
```

Nested Try Catch Example

Here we have deep (two level) nesting which means we have a try-catch block inside a nested try block. To make you understand better I have given the names to each try block in comments like try-block2, try-block3 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` 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 gets transferred to the main (grand parent) try-block where it found the

appropriate catch block for exception. This is how the the nesting structure works.

Example 2: Nested try block

```
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..
```

This is another example that shows how the nested try block works. 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 ArithmeticException, since the catch of block1 is handling ArithmeticException `"Exception: e1"` displayed.

Block2: In block2, `ArithmeticException` occurred but block 2 catch is only handling ArrayIndexOutOfBoundsException so in this case control jump to the Main try-catch(parent) body and checks for the ArithmeticException catch handler in parent catch blocks. Since catch of parent try block is handling this exception using generic Exception handler that handles all exceptions, the message “Inside parent try catch block” displayed as output.

Parent try Catch block: No exception occurred here so the “Next statement..” displayed.

The important point to note here is that whenever the child catch blocks are not handling any exception, the jumps to the parent catch blocks, if the exception is not handled there as well then the program will terminate abruptly showing system generated message.