

FUNDAMENTALS OF OBJECT ORIENTED PROGRAMMING

Lecture28

Lecture 28: Multiple Catch and Nested try Statements


Welcome to lecture number 28. Exception handling. So, in the previous lecture, I talked about the try-throw-catch with respect to Java, all right. So now, what will happen if there is a situation where there are multiple, let us say, catch blocks, all right? So, you may have more than one exception, all right.

So, till now, we discussed one exception. So, what will happen, right? So, if there are more than one exception, right, raised by a single code. So, if you want to handle These cases, right, this kind of situation, we have to specify two or more catch clauses, right?

So, there may be an array out of bounds or there may be an arithmetic exception. So, in that case, what do we have to do? So, we have to specify two or more, right? So, these exceptions, right? So, each catching exception should be of a different type.

Multiple catch Clauses

- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you can specify two or more catch clauses, each catching a different type of exception.
- When an exception is thrown, each catch statement is inspected in order, and the first one whose type matches that of the exception is executed.
- After one catch statement executes, the others are bypassed, and execution continues after the try/catch block.
- The example given in next slide traps three different exception types:



Right. So that is what I said. It may be arithmetic. It may be array out of bound or when you are converting from string to integer. Right.

So you have to specify the exception. And moreover, so when you are writing the catch statement. Right. Let us assume. Right.

At one point of time, we have to write because the exception will be thrown. And when you are writing the catch statement. So that has to follow some order. Right. That has to follow certain order.

Like, let us say, arithmetic exception will be at the top. It is like a superclass, and then the subclass will follow. So, suppose if there is one catch statement that executes. Right. Assume that the one catch statement is getting executed.

Then the others are bypassed. And execution continues after the try-catch block. Right. So this is similar to the one catch. Right.

Similar to the one catch. So, suppose let us consider. Three different exceptions, right? That we are going to see in the next slide. So, three different exceptions.

So now, you have triblock. You are creating an array of five elements, a dynamic array of five elements in Java. And then, what you are doing—you are doing two things, right? One is on the right-hand side; if you look, it is 100 by 0. Right.

Multiple Catching in JAVA

```
1- public class TestMultipleCatchBlock{
2-     public static void main(String args[]){
3-         try{
4-             int a[]=new int[5];
5-             a[5]=100/0;
6-         }
7-         catch(ArithmeticException e){System.out.println("task1 is completed");}
8-         catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
9-         catch(Exception e){System.out.println("common task completed");}
10
11         System.out.println("rest of the code...");
12     }
13 }
```

When you look at the right-hand side, it is 100 by 0, and on the left-hand side, it is A of 5. So you can access only A0, A1, A2, A3, and A4. Right. So when you try to access A5, there is an error. Now, the catch is right.

In fact, the catch. So which catch block will be called, or which catch function will be called? So try. There are two exceptions. One is 100 by 0.

And then the left hand side A of 5, you are trying to access 5th index. You can access only 0, 1, 2, 3, 4th index, maximum 4th index. The question is now which catch, right, will be executed, right? So, here you are seeing line number 7, you have arithmetic exception. And line number 8, you have array index out of bounds, right?

And you have the exception, right? So, you have one exception, well defined with E, right? You have an exception, right? So there are three exceptions are there. So now you can think of which one will be executed.

This is an interesting point. Actually, so if you look, the arithmetic exception is like in the superclass. Rest of them are in subclass. And moreover, we have to maintain the order, right? We have to maintain the order.

A practical example I will give in the next slide. Alright. So, here you can see the arithmetic exception is at the top. Next is array index out of bounds and then exception. So, there will be some hierarchy.

Right. There is a hierarchy. So, arithmetic exception will go to the top because it is like a superclass. The rest of them are subclasses. So, what will happen here?

It will catch line number 7. Some of you may say that it will catch Right. I mean, seven and eight. You can say line number seven and line number eight will be executed.

Right. Hundred by zero ones and a five next. Right. It is not. Suppose in cricket.

Right. Let us say the batsman is eating, and who will be catching the ball? Only one fielder is catching the ball. Correct. So, in a similar way.

When you have three catches, I am saying the function, multiple catches, right? You have a multiple-catch function, but only one will be executed, right? So, when I am hitting, only one fielder is going to catch the ball, correct? So, in a similar way, here the arithmetic exception is at the top. So, since you have the arithmetic exception at line number 5, right?

So, it will throw the exception first. All right. And then it will be caught by line number 7. So, 'Task 1 is completed' will be printed, and that is it. Line numbers 8 and 9 will be ignored, and it will go to line number 11.

The rest of the code will be printed. This is what exactly happens. All right. So, if I run this code, you can see that 'Task 1 is completed'. That means 100 divided by 0, which is the first preference in the hierarchy.

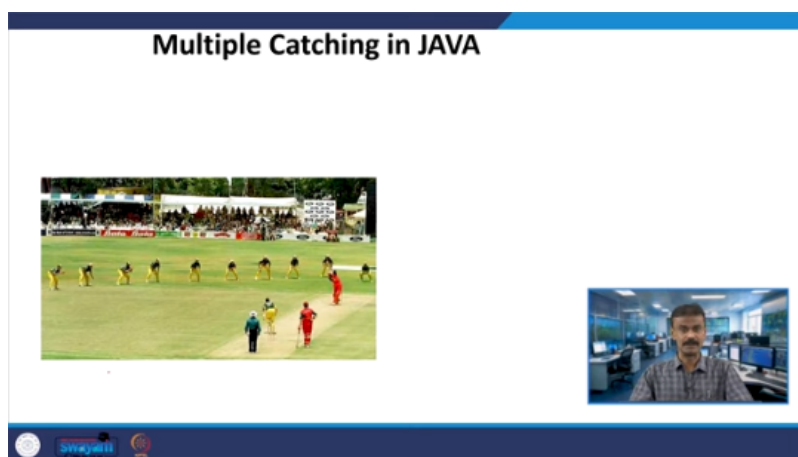
All right. You might have studied the hierarchy of operations when learning the basics of procedural-oriented languages. So, in a similar way here, when you have multiple catch blocks, the catch also follows a hierarchy. So, since I have here 100 divided by 0.

So, the first hierarchy I have to write is arithmetic exception. So, that will be caught, and then you look, task 1 is completed, is printing. That is it. Right. The ball has been caught.

Batsman is out. Right. So I do not need to worry about line numbers 8 and 9. Right. Exactly, this is happening.

Then it is coming to line number 11. Rest of the code will be printed. So when you compile and run the code in terminal, you can see that task 1 is completed and rest of the code will be printed. So now, as I said. Right.

So here you see 9 filters in one frame. So, when the batsman hits the ball, you know that only one fielder will catch it. So, this is exactly what you should remember from the previous example. Also, there is a hierarchy. If you take the Indian test team, I mean, I do not know how many of you know, there is a hierarchy among the slip fielders.



You can see first slip Virat Kohli, second slip KL Rahul, and third slip Rohit Sharma. That is a hierarchy. Right. You can see, or in earlier days, Rahul Dravid was in the first slip. Right.

And then maybe Sachin Tendulkar. Right. And then Saurav Ganguly or V.V.S. Laxman. So you can see the hierarchy in the first slip, second slip, and third slip.

So, in a similar way, if you take the previous code. Arithmetic exception as a hierarchy than the array index out of bound. And then the exception is going. The third one is exception is going. So, you should remember that first slip, second slip, and third slip.


So, that is why I put this. Alright. And then, so when you are having the exception. So, at a time, only one exception will occur. For example, in the previous code, right-hand side you have 100 by 0.

So, right-hand side you have 100 by 0. So, that is arithmetic exception. So, then one catch block will be there. So, it is going over there. And then, that particular catch block is being executed.

Fine over. The ball has been caught. The batsman is out. So then it is going to line number 11. Line number 7, it has been caught.

Multiple Catching in JAVA

- At a time only one Exception is occurred and at a time only one catch block is executed.



And then, The flow is there because it will not go further. Catch functions, right? Some of you may say line number 5, you have an A of 5, right? Array index out of bounds. So, will it be caught, right?

It is, I mean, not, right? It is, the answer is not, correct? So, once the batsman is out, he has to go out, right? So, line number 7, it has been caught. Fine over, it will go to line number 11.

So, All the catch blocks must be ordered from most specific to most general, right? So, it has to follow a certain hierarchy. The slip fielders example I gave works similarly because we are talking about multiple catches. So, in this case, the arithmetic exception must come first, right, before any other catch exceptions you are using, right?

In the previous example, we used arithmetic exception first, then array out of bounds, and then the exception, right? Right. So, there is an order. You can see what the order is—it follows like one is a superclass. In fact, arithmetic exception is a subclass, and then there are subclasses. We have already seen how inheritance works, right? So, in a similar way, it goes to the arithmetic exception, and then it will work. So, for example, right, so this

Picture I took from this particular website. And if you look at this code, right? The same code, right? So, AF5 is 30 by 0. So, I have an arithmetic exception.

And then, on the left-hand side, I have AF5, right? I swap the slip fielders, right? Now, first slip has gone to second slip. Second slip has gone to third slip. And third slip has come to first slip.

I have changed the hierarchy. Is it possible, right? So, if you look at any cricket team, right? Any test match cricket team. They have the hierarchy in the slip field.


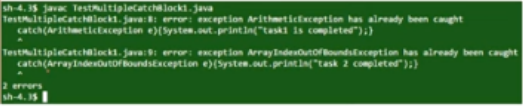
Suppose this happens. All right. So, I have not changed the code. I only tried to change the order of the catch function. But unfortunately, this is not possible.

All right. So you will get an error. All right. So there are two errors, saying that the first error is: 'Exception ArithmeticException has already been caught.' All right.

So in fact, a '30 divided by 0' error. It has already been caught, but unfortunately, you have put line number 7 before. Right? Similarly, the 'ArrayIndexOutOfBoundsException' error for index 5 has also been caught. But your hierarchy is Exception.

Multiple Catching in JAVA

```
1- class TestMultipleCatchBlock1{
2-     public static void main(String args[]){
3-         try{
4-             int a[]=new int[5];
5-             a[5]=30/0;
6-         }
7-         catch(Exception e){System.out.println("common task completed");}
8-         catch(ArithmeticException e){System.out.println("task1 is completed");}
9-         catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
10        System.out.println("rest of the code...");
11    }
12 }
```



So therefore... If you look at this particular code and run it in the terminal, you will get errors. In fact, you are getting two errors. So, you cannot change the hierarchy of the exceptions. So, from a Java perspective, when talking about multiple catches, I cannot change the hierarchy.

I cannot change the hierarchy of multiple catching. So, first I have to put arithmetic exception. Line number 8 should be in line number 7. And line number 7 should be line number 8, right? And then exception, the last one, line number 7 should go to line number 9.

You just change. You just make a change. Make line number 7 and put at the last. And then try to execute the code. In fact, we have done that.

It will run, right? So, when you have multiple catching, so this is the rule you have to follow. And then you can see. So this I am giving as an assignment. So see what are all the different exceptions.

Like, I mean, in cricket, you can put 9 slip fielders. Assume that you have 9 or 20. Find out what all the exceptions are in Java. And then you see what all the orders you have to follow. I hope it is clear for everyone.

So if I change, you will get an error. If I change the hierarchy, you will get an error. So caution on multiple catching. As I already said. You have to follow a certain order.

So this is due to technicality. So your subclass. Must be placed. Before any of its superclass. In a catch block.

Caution on Multiple Catching

- Keep in mind that an exception subclass must be placed before any of its superclasses in a catch block.
- This is because a catch block that catches a superclass will handle exceptions of that type as well as its subclasses.
- Therefore, if a subclass comes after its superclass, it will never be reached.
- For instance, `ArithmeticException` is a subclass of `RuntimeException`. Additionally, unreachable code in Java results in a compilation error.



So, in fact, we have talked about inheritance. The superclass and subclass properties. So, whenever the catch block catches the superclass, the superclass will handle

exceptions of the type as well as its subclasses. So, suppose if a subclass comes after its superclass, it will never be reached. So, for instance, `ArithmeticException`, which we have seen, is a subclass of `RuntimeException`, right?

And also, unreachable code in Java results in a compilation error. That is what we got, right? So, I try to put `Exception` first, then `ArithmeticException`, and then `ArrayIndexOutOfBoundsException`. So, then we can see that Right.

So the code is unreachable. And in that case, you may get a compilation error. So the main idea is you have to follow a certain hierarchy. This is due to the properties of subclass and superclass. Similarly, you may ask the question: can I have several try statements?

Nested try Statements

- A **try** statement can be nested within another **try** block, meaning one try can exist inside another.
- When an exception occurs, the context of the exception is placed on the stack.
- If the inner **try** block does not have a matching catch handler for the exception, the stack is unwound, and the next enclosing **try** block's catch handlers are checked for a match.
- This process continues until a matching catch is found or all nested **try** blocks are exhausted.
- If no match is found, the Java runtime system will handle the exception.



You call them nested try statements. Yes, in Java, you can have several try statements. All right. So you will have one try block, then a second try block. All right.

And then third, fourth, and so on. All right. So in that case, suppose when an exception occurs. All right. So the context of the exception, you are going to place in a stack.

Right. For example, I'm putting the first try in a stack. That means in the complete block. And whenever the catch occurs, assume the second is also try, then catch occurs, they immediately catch and try, right? So try, the exception occurs, that catch, particular catch block will be caught, right?

So this is what exactly happening. So whenever you have the try block, put it in a stack, right? In fact, it is happening in the stack. It is placing in a stack. Correct.

And then whenever the catch block comes. Right. And assume that inside the try block, I have exception. So the immediate catch block will be called and then it will be executed. Right.

If you suppose you may ask the question, there is no match form. So, match form, see before try-throw-catch, what did you do? Assume that you have written a code: a is equal to 50 by 0. Correct. So then what happens?

So, in the default, right, the standard library, whether C++ or Java, it will be called. So that is why, if you look at the last point, if there is no match found, the Java runtime system will handle the exception. It is exactly like before studying this exception; automatically it will be called. Suppose in my program I have age equal to 50 by 0, assume that I am not using try-throw-catch. Correct?

So, you know that this will give a runtime error. So, a similar thing will happen if there is no match. Right? So, for example, here. Right?

So, assume that I have this here. Right? So, the first one I am getting. This is my first try. Right?

So, I will put try one. I will call it try one. Right? I will put it in the stack. Correct?

```
....
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
....
```

Syntax

try 1

So, the stack is nothing but it will work on the first in, last out or last in, first out principle. Right? It is like a file. In the office, right? Assume that the officer is coming around 11 o'clock, and the subordinate is putting the files. The first file goes at the last.

Then second file, third file. So, the files will be piled. The first one he has put will be at the last. That is the physical example of a stack. Alright.

So, here in the stack, in the stack memory, I put the first try. And then you have second try. Right. Nested. This is called the nested.

So, I will put the second try too. Correct. So, it is executing. It is keep on executing. So, assume that I have.

Right. So, this is second one. So, assume that I have some exception here between the statement 1 and statement 2 or one of the statements, right? So, then what happens? This is the catch 1 you are inserting into the stack, pushing into the stack, right?

So, catch 1, right? And then you will have another catch, catch 2. So, what will happen when there is some exception? Assume that statement 2, I have exception, right? So, try 2.

Syntax

```

....
try
{
    statement 1;
    statement 2;
    try
    {
        statement 1;
        statement 2;
    }
    catch(Exception e)
    {
    }
}
catch(Exception e)
{
}
....

```

Handwritten annotations on the slide:

- A circled '1' points to the outer try block.
- A circled '2' points to the inner try block.
- A circled '3' points to the inner catch block.
- A stack diagram labeled 'FILO' and 'LIFO' shows a stack with 'try 1' at the bottom and 'try 2' on top.
- A small video inset shows a man speaking.

So, what will happen? So, this is catch 2, right? So, try 2. So, you can see the immediate catch, right? So, these two will be executed.

So, try. If there is any exception, for example, assume that I am finding a is equal to 100 by 0 here. right, a is equal to 100 by 0, then that means this particular catch, all right, this particular catch will be executed, this exception will be executed, right, so this is over, right, so now the pair is try 1 and catch 2, right, so when this is happening, right, in from this try, this is a block, complete block, block goes from here to here, right, so in case, Correct. So in case any exception happens.

Right. So in case any exception happens. So these two will be executed. So try one. Suppose here try assume that this particular statement is having some exception.



All right. So that will be caught by this. right. So, this hierarchy right. So, this will be followed correct.

This will be followed. So, in the case when you have nested try and catch. So, let us consider this example. I have two tries, right? So, here I have an exception, right? The class exception 6.

So, let us have The main program: int c is equal to 100. Alright. So, this is my first try 1. Right.

I put this try 1. Right. I put this try 1. And then I will have another try. Try 2.

```
1 class Excep6{
2 public static void main(String args[]){
3     int c=100;
4     try{
5         try{
6             System.out.println("going to divide");
7             int b =39/0;
8         }catch(ArithmeticException e){System.out.println(e);}
9     }
```



Correct. Try 2. Anyway, I am getting the exception over here. So, this try 2, you have one catch, right? So, here there is no pair.

You do not have one more catch, right? So, try 2, this is try 2, correct? I have already named try 2. So, there is an exception here, 39 by 0, right? So, 39 by 0, and it is trying to assign B, right?

There is an exception. So, that means try 2 catch 1, right? All right. So, that means this will be executed for this try. This will be executed.

Right. So, this will be executed. And fortunately, we do not have any other exceptions. Right. Even if there is no match.

This is what I said. Suppose there is some more coming. Let us assume. Correct. So, in that case, what will happen?

See, automatically, a runtime error will occur, right? But fortunately, in this case, right? So, what is happening? So, arithmetic exception E. So, we are printing System.out.println E. Maybe the program is continuing. Maybe we will see what the message is, etc., right?


So, catch. So, we have one more try, right? So, line number 10, you have one more try. So I'll call it as try 3. I'll call it as try 3.

We have one more try and maybe I have to draw slightly more. What I will do? I'll try here. Try 1 and then you have try 2 and you have catch 1 and then your try 1 is coming. Right.

Try 3 is coming. Right. And then I have certain catches. Correct. So here I have catch 2.

This I call try 3. And this is catch 2. Catch 2. And finally, I have one catch 3. Correct.

```
10 * try{
11     int a[]=new int[5];
12     a[5]=4;
13 }catch(ArrayIndexOutOfBoundsException e)
14 {System.out.println(e);}
15
16 System.out.println("other statement");
17 c=c/0;
18 }catch(Exception e){System.out.println("handeled");}
19
20 System.out.println("normal flow..");
21 }
22 }
```



So now I will rub this part. Right. So let us consider this particular stack. Now I hope you understand. Right.

So let us consider this particular stack. So, now what is the pair? So, you have try 2 and catch 1—that immediate pair I am taking, right? So, try 2 and catch 1. So, arithmetic exception—that is, divide by zero arithmetic exception, right?

So, try 2 catch 1, right. So, next you have try 3 and catch 2, the immediate one, right. So, try 2. 3 and catch 2. So, try 3 is what?

Try 3 is correct. So, that is an array index which is going out of bounds exception. So, try 3 catch 2. So, try 3 catch 2, the immediate one, right? Try 3 catch 2, and finally try 1 catch 3. Try 1 catch 3. So, this will be your catch 3.

So now you have try, so you will have an array index out of bounds. So you are printing System.out.println(e). So whatever the exception is, it will be printed. Similarly, line number 8, whatever the exception is, that e will be printed. So that is what we have to understand. System.out.print(l and e). So that arithmetic exception, whatever the exception that belongs to arithmetic exception, that will be printed.

So 2 1 done. Then I will go for try 3 and catch 2. So try 3 and catch 2. So here you have array out of bound. So whatever array index out of bounds.

corresponding exception will be printed and then it is printing other statement. So I do not have a so first I will print going to divide. So then arithmetic exception will be printed right. So then array index out of bound will be printed. Then other statement will be printed right.

So here you are having C is equal to C by 0. So, this is also over try 3 catch 2. So, try 3 over here line number 10 and catch 2 is obviously array index out above. Line number 17 your C is equal to C by 0. What is C?

Int C equal to 100 line number 3. So, 100 by 0 exception. So, try 1 will catch 3. What is catch 3? This one exception system dot out dot println you put handle will be printed.

So, then you will have normal flow, correct? So, then you will have normal flow. So, when I run the code, so going to divide first, so that is what we have written, this is printed. Second one is you are printing E, which is nothing but arithmetic exception, what is the statement?

So, this will be printed divided by 0, right? And then array index out of bound, right? Index 5 out of bound for length 5, yes, it should be 0, 1, 2, 3, 4, right? So, this is printing, array index out of bound.

Finally, so here you are printing handled, right? So, other statement will be printed in line number 16 that is printed and then you are printing is you are not printing E, right? You are printing this particular statement system dot out dot print all and handle when the exception occurs C is equal to C by 0 you have handled, right? So, that is being printed and then the normal flow it is going line number 20. Right.

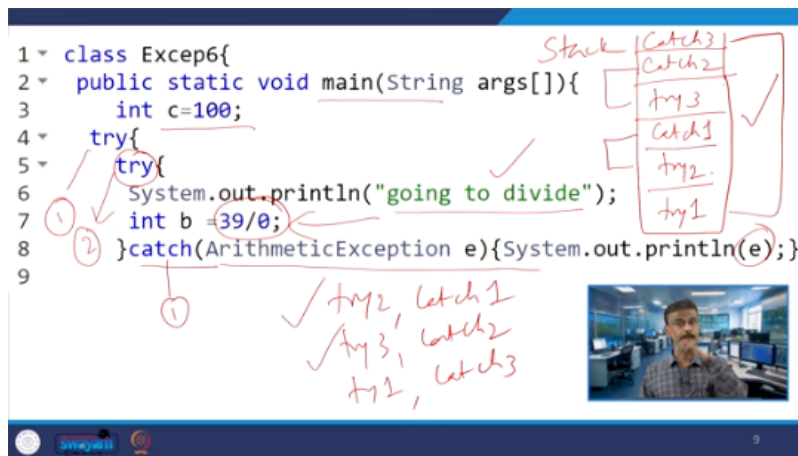
Even though you have three exceptions, still it is going till line number 20 normal flow. So this is a very good example. Right. So you can look because you are learning two concepts. One is you have multiple try and catch.

So that means you have three exceptions. One is at line number seven and second one is line number 12 and the third one is line number 17. And then you have multiple catches. So, where exactly it is being caught, right? So, for this,

From the stack. So, try 1. Right. It is like popping up catch 1 and try 2. Right.

Popping out catch 1 and try 2. Then you are pushing try 3. Right. And catch 2

```
1 = class Exception {
```



So try 3 you are getting array out of bound. Right. Line number 12 you are

And then finally you have catch 3. So the try 1 and catch 3 will be the pair. So


And then all these examples we have seen, they are throwing exceptions, right?

this, can we use it in Java, right? The answer is yes, which we will see in the next lecture. Thank you.

```
10 try{
11     int a[]=new int[5];
12     a[5]=4;
13 }catch(ArrayIndexOutOfBoundsException e)
14     {System.out.println(e);}
15
16 System.out.println("other statement");
17 c=c/0;
18 }catch(Exception e){System.out.println("handeled");}
19
20 System.out.println("normal flow..");
21 }
22 }
```

Handwritten annotations on the slide:

- A red circle with the number 3 is next to line 13, with an arrow pointing to the `ArrayIndexOutOfBoundsException` exception.
- A red circle with the number 2 is next to line 15, with an arrow pointing to the `System.out.println("other statement");` line.
- A red circle with the number 3 is next to line 22, with an arrow pointing to the closing brace of the `try` block.
- A red checkmark is next to line 18.
- A red arrow points from the `handeled` string in line 18 to the `try` block.



10