Java Multi-catch block

We've learned about try/catch blocks–but we can easily take our error handling to another level. For instance: what if there are a bunch of potential exceptions that could happen within a single try block? It'd be nice to handle each exception with its own solution, right? Luckily, we can. Indeed, a `try` block can be followed by as many catch blocks as we need, each with its own appropriate error-handling code.

#Which Error?

Hopefully, you remember if/else statements and how to string them together:
CopyC#C++ClojureCSSDartGoHaskellHTMLJavaJavaScriptJSONJSXKotlinMarkdown
PascalPerlPHPPlain
TextPythonRRubyRustSchemeShellSQLSwiftTypescriptVB.NETVBScriptXMLYAML

```
if(condition1) {
// What to do if condition1 is met
} else if(condition2) {
// What to do if condition2 is met
} else if(condition3) {
// What to do if condition3 is met
} else {
// What to do if none of the conditions is met
}
```

When learning to code, you're going to learn a lot of syntax, and you're going to have to get it right to make a program run. However, to be a true programmer, you have to spend at least a little of your time focusing on the patterns that govern code like this. Although you're only learning Java at the moment, the `if else if` statement above is a common fixture in many languages, and you'll be seeing it again. The try-catch block is everywhere, too, probably because of the extremely basic pattern that makes it work.

Consider the pattern above, then try the short answer question below:

Describe the `if else` pattern above in your own words, then think about exception handling with try/catch/finally. Which statements/parameters are analogous (aka, the same)? What do you think will have to be different in each `catch` in order to have multiple `catch` blocks attached to a single `try`? **Don't scroll down until after you've answered: take the time to think it through.**

The if-else pattern presented above is a series of conditional checks where the program evaluates each condition sequentially. If a condition is met (evaluates to true), the corresponding block of code is executed. If the first condition is met, its associated code block executes; otherwise, the program moves to the next condition and so on. If none of the conditions are met, the final else block is executed.

In the context of exception handling with try-catch blocks, the try block contains the code that might throw an exception. If an exception occurs in the try block, the program "catches" the exception and executes the code within the corresponding catch block that matches the type of the thrown exception. If there are multiple catch blocks, the program will check each catch block sequentially to find the appropriate match for the thrown exception type. Once the matching catch block is found, its code is executed, and the other catch blocks are skipped. If no matching catch block is found, the program can throw the exception to an outer scope or terminate.

Once you've submitted your answer above, reveal the hidden content below and check to see how well you did:

Reveal Content

InfoWarningTip

Make sure you've filled in the short answer question above before you scroll down. Remember: learning to code is not about **memorizing** steps. It's about learning how to understand patterns, predict them, and use them no matter what language you're asked to write in. If you're still taking a look at documentation, going back to previous lessons to refresh your memory, or looking at a cheatsheet–that's fine!

You'll continue to do so throughout your career–but always do your best to talk it out before you move on.

Here's what a multiple try/catch block looks like:

CopyCC#C++ClojureCSSDartGoHaskellHTMLJavaJavaScriptJSONJSXKotlinMarkdown PascalPerlPHPPlain TextPythonRRubyRustSchemeShellSQLSwiftTypescriptVB.NETVBScriptXMLYAML

```
try {
// The block of code we'd like to accomplish
} catch (SpecificBuiltInException e) {
// What to do if the exception we've specified in the parentheses occurs
} catch (SpecificBuiltInException2 e) {
// What to do if the exception we've specified in the parentheses occurs
} catch (Exception e) {
// Generic catch for all other errors using the built-in Exception class
} finally {
// Code to run no matter what the outcome of the try/catch blocks above were
}
```

#Order Matters

We'll talk about writing our own custom exceptions later. For now, the specific exceptions we're talking about will be ones that are built into the Java programming language.

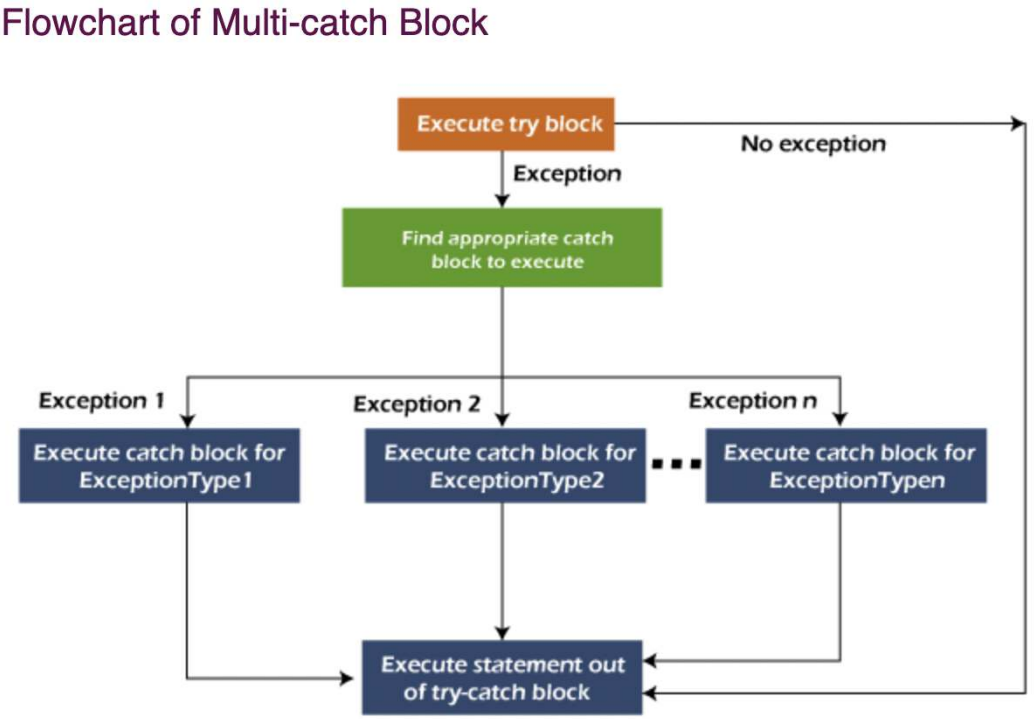
More importantly: their order will matter. Let's watch an example in action, making sure to pay special attention to the ordering and the explanation of how the catches run at the 5:40 mark:

Open in new tab

In your own words, why does the ordering of our `catch` exceptions matter? What sorts of problems might occur if you're not careful? How might this affect your code/your user's ability to properly utilize your program?
The ordering of catch exceptions matters because in exception handling, catch blocks are evaluated sequentially from top to bottom. When an exception is thrown, the program checks each catch block in the order they are written to find the first catch block that matches the type of the thrown exception. Once a match is found, the code in that catch block is executed, and the remaining catch blocks are skipped.

#Points to Remember:

- Each `catch` block should be responsible for a specific exception, except for the last one which should catch exceptions more generally.
- All `catch` blocks must be ordered from **most specific to most general**, i.e. the `catch` for `ArithmeticException` must come before `catch` for `Exception`, otherwise none of the specific `catch` blocks further down the line will ever run.
- No matter how many catches you have, you can always have a `finally` block to wrap up your try/catch code, if you want.
- Exception types may need to be imported, but you can often use them without importation. Try your code, see if it runs. If not, import as necessary–good IDEs like Eclipse should underline code if it's not working, then fill in the import for you with a couple of clicks. Get used to this pattern!



A flowchart showing how exceptions are run when we use multiple catch blocks.

#Here's an example:

Take some time to play around with the example below. Use exception types from previous lessons, cause some failures, and see if you can iterate on what we've written out for you. The YouTube video above does a good job showing how to figure out what kinds of exceptions can occur in a file like this, so feel free to change the setup to take in some numbers and see what you can figure out. We recommend you spend at least ten minutes in this sandbox before continuing, but we'll leave the decision up to you:

Run

MultipleCatchBlock1.java

```
public class MultipleCatchBlock1 {

    public static void main(String[] args) {
        try {
            int a[]=new int[5];
            a[5]=30/0;
        } catch(ArithmeticException e) {
            System.out.println("An Arithmetic Exception occurred.");
        } catch(ArrayIndexOutOfBoundsException e) {
            System.out.println("An ArrayIndexOutOfBounds Exception occurs");
        } catch(Exception e) {
```

```
            System.out.println("An Exception somewhere in the parent object oc
curs");
        }
        System.out.println("The rest of our code can continue as usual");
    }
}
```

#Knowledge Check

Put the exception handling code in the correct order. Note: not all blocks in this question type will always be used. Some may be distractions from the actual solution.

Options

```
} else {
} else if(condition2) {
if(condition) {
```

Solution

```
try {
// Code
} catch (SpecificBuiltInException e) {
// Code
} catch (SpecificBuiltInException2 e) {
// Code
} catch (Exception e) {
// Code
} finally {
// Code
}
```

Fill In The Blank

Because they're checked in order from top to bottom, catch blocks must be ordered from most to most in order to work properly.

#Extra Resources

More examples :https://www.javatpoint.com/multiple-catch-block-in-java

And if you'd like to see a huge list of possible errors and the packages they come from, you can check one out here.