Rex
Jones II

How To
Handle Exceptions

Table of Contents

Rex Jones II

Intro

Next, is How To Handle Exceptions. Before talking about How To Handle Exceptions. Let's talk about

Why We Should Handle Exception

"Why we should handle exceptions?". Recall from the introduction, an exception is an indication that something is wrong with our program. Perhaps the logic is wrong and we need to correct it or throw an exception. Catch an exception. Let's go to Eclipse.

Start with the main method and the value. int value = 34/0. Print the value. System.*out*.println("Value: " + value); We are unable to divide any number by zero so 34/0 is not a valid operation. The syntax is correct so there is no problem when running our program.

Notice, the program terminated without printing a value. However, we see an exception. Read the exception from bottom up. The package is com.selenium4beginners.java.exceptionhandle, class is NoExceptionHandling, main is the method. NoExceptionHandling dot java colon 7 shows the class name and line number with a problem. The previous line is the Exception showing thread "main" java.lang.ArithmeticException : / by zero. This mean there is a problem with our logic. Click the line number and we are directed to the root of our problem.

How To Handle An Exception

How To Handle An Exception? We handle exceptions using 4 keywords and a Test Framework: try, catch, TestNG, throw, throws, and finally.

Ways To Handle Exceptions

try and catch are the core of handling exceptions. Try is used to monitor code that may cause an exception. Catch is used to handle an exception if an exception is thrown. We can choose to add more than one Catch statement to handle an exception. What's the purpose of handling an exception? The purpose is to give power to our code. With that power, our code can respond to any problem then continue running our program. Let's take a look.

Demo / Try-Catch

Copy the previous code. Write try – CTRL + SPACE. Copy the code and place it within the brackets for try. Exception is the type of exception and e is the object that stores a value after catching an exception. However, I'm going to change Exception to ArithmeticException and e to exc. I changed Exception to ArithmeticException because ArithmeticException is the type of exception that happens after dividing by zero. Add a print statement sysout ("Can't Divide By Zero"). We can also call some methods by writing exc, the object and the dot operator and some methods are getLocalizedMessage, getMessage, getStackTrace but I'm going to leave off calling an exception. Let's Run. Can't Divide By Zero. This is how we handle exceptions.

In this example, we know the exception is ArithmeticException.

Rex Jones II

Demo / Try-Multiple Catch

Copy this code and paste the code. If we are not sure of the exception then it is common to add multiple catch statements. Each catch statement must be different. Let me show you the superclass which is Exception since it deals with all program related exceptions. Write
catch (Exception e)

```
{
  System.out.println("There Is A Problem");
}
```

and run. We see the same print statement. Can't Divide By Zero. If we use Exception then it must be placed in the last catch statement. Look what happens when we add Exception before ArithmeticException. "Unreachable catch block for ArithmeticException. It is already handled by the catch block for Exception". Everything after the superclass Exception cannot be reached. Change it back.

TestNG

We can also handle exceptions using TestNG. There's an attribute called expectedExceptions for the Test annotation. All we need to do is pass in 1 or more classes. The purpose is to list exceptions that a test method is expected to throw.

throw

So far, up to this point, we have caught exceptions created automatically by the JVM. The throw statement makes it possible to manually throw our own exception. We write keywords throw new followed by the name of the Exception class.

throws

A method must declare an exception in the throws clause if the method generates an exception that it does not handle. However, the Throwable subclasses Error and RuntimeException are not required in the throws list. All other exceptions will show a compilation error.

finally

If there is an exception or not an exception, the finally block of code will always execute regardless of the condition. Let's go back to Eclipse.

Demo / finally

Say for example, we have a database connection, network connection, or maybe a file that's open. At the end of the try-catch block, we can add a finally block and a print statement. sysout("Close The Open File"). This block of code will always execute. Let's Run and we see Can't Divide By Zero. Close The Open File. Remove the exception by changing 0 to 2. Let's run again. There is no exception. Value equals 17. Close the open file. The print statement for finally also showed up. That's it for how we handle exceptions.