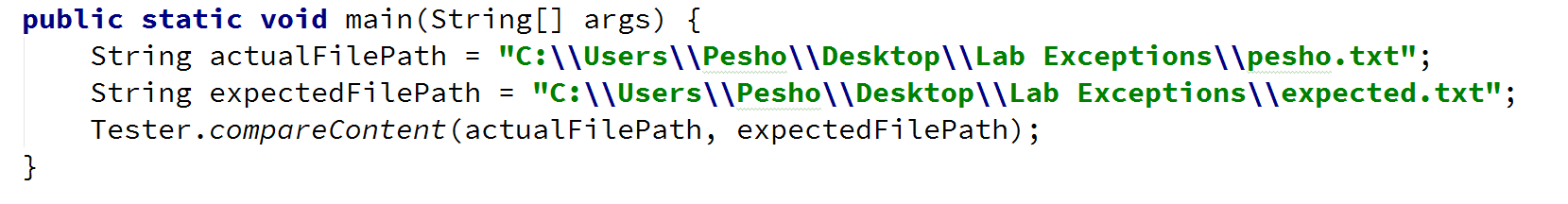
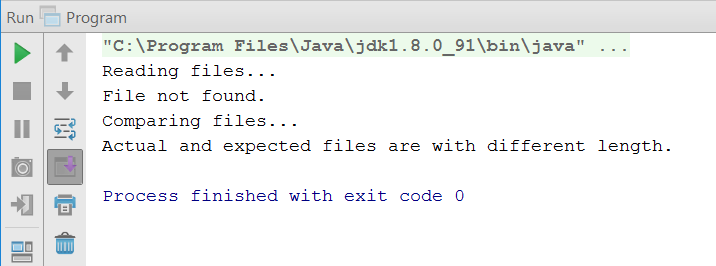
# Lab: Exception Handling

## Reading a Missing File

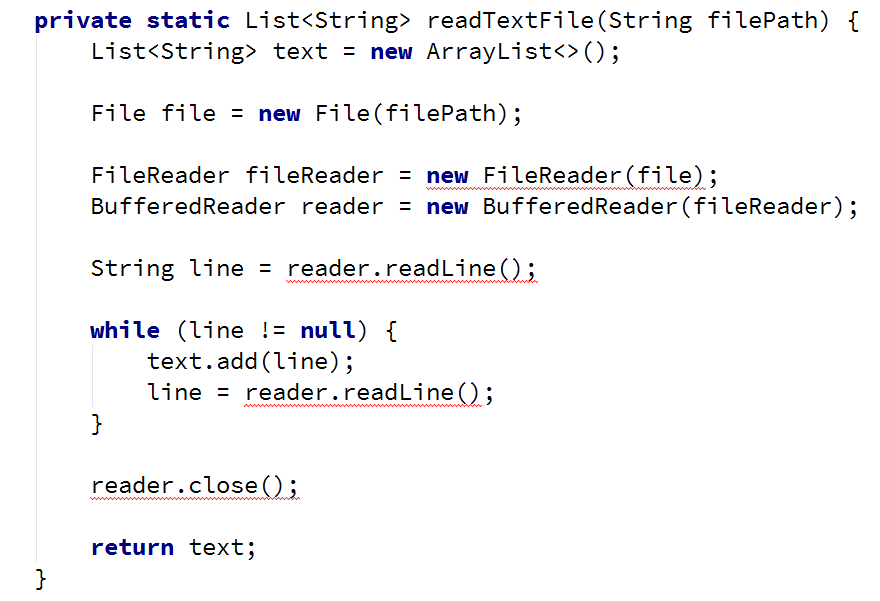
By now you should know that you can either handle or throw an exception and the Java language protects you by making this mandatory. Now let’s consider the readTextFile() method in our Tester class. Previously, while reading text files we did catch an exception and handled it within the method. But what would happen if you test it with a missing file? You will see that the code executes, you get an error message and then the program continues forward. Let us test this right now. Try testing with the following code in your main method and of course edit the file paths to ensure you are passing one existing file and one that does not:



Unless some unusual reason you really shouldn’t have a file called pesho.txt. So if you try the above code you should get an Exception with the message “File not found.” because that is the way we handle it right now. But the code that follows will still execute and you will get an unexpected output:



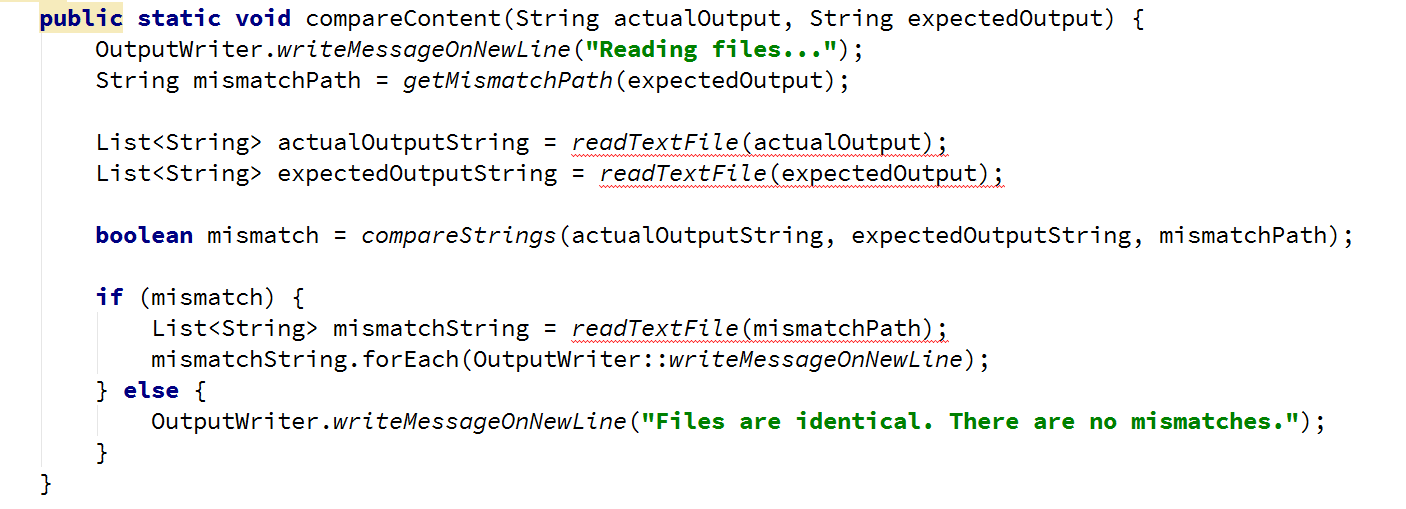
Well if you don’t find one of the files what is the reason to still compare them. Let’s try to fix this. First we will remove the try-catch block from the readTextFile() method:



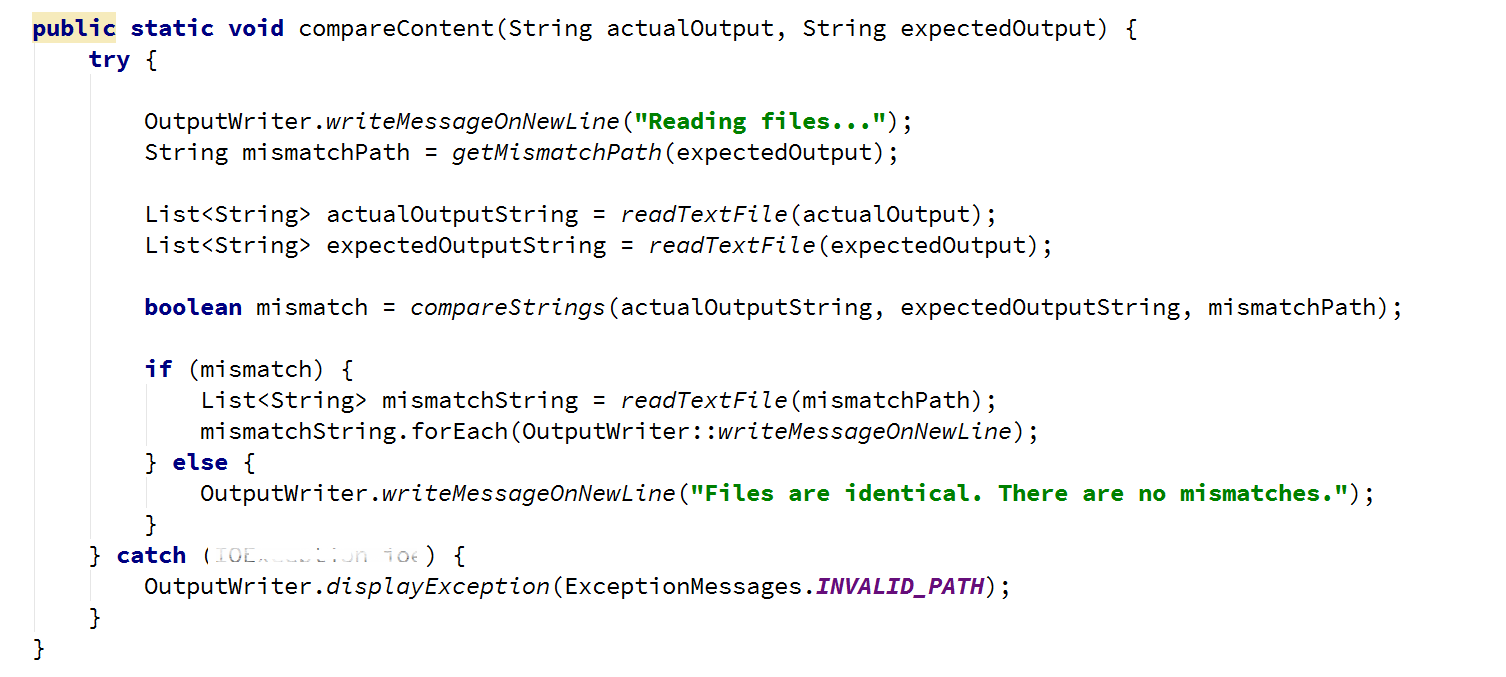
And immediately you can see that this won’t compile. Java wants us to do something with the potential exception. We can just make our method throw it so we add “throws IOException” to the methods signature:



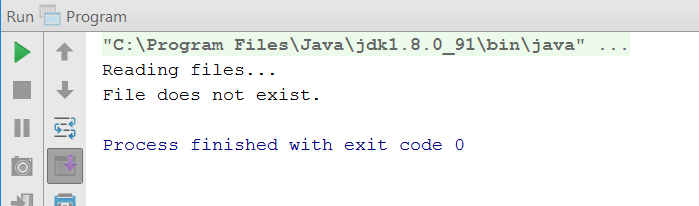
But now we can see that we broke something else. In compareContent() method Java is telling us that we have an unhandled exception. If you look at your code you should see this:



So in the compareContent() method we must catch the exception that we threw in readTextFile() and display a proper message, which we already have. We can use the INVALID\_PATH message that we created earlier. Ok, let’s do just that:

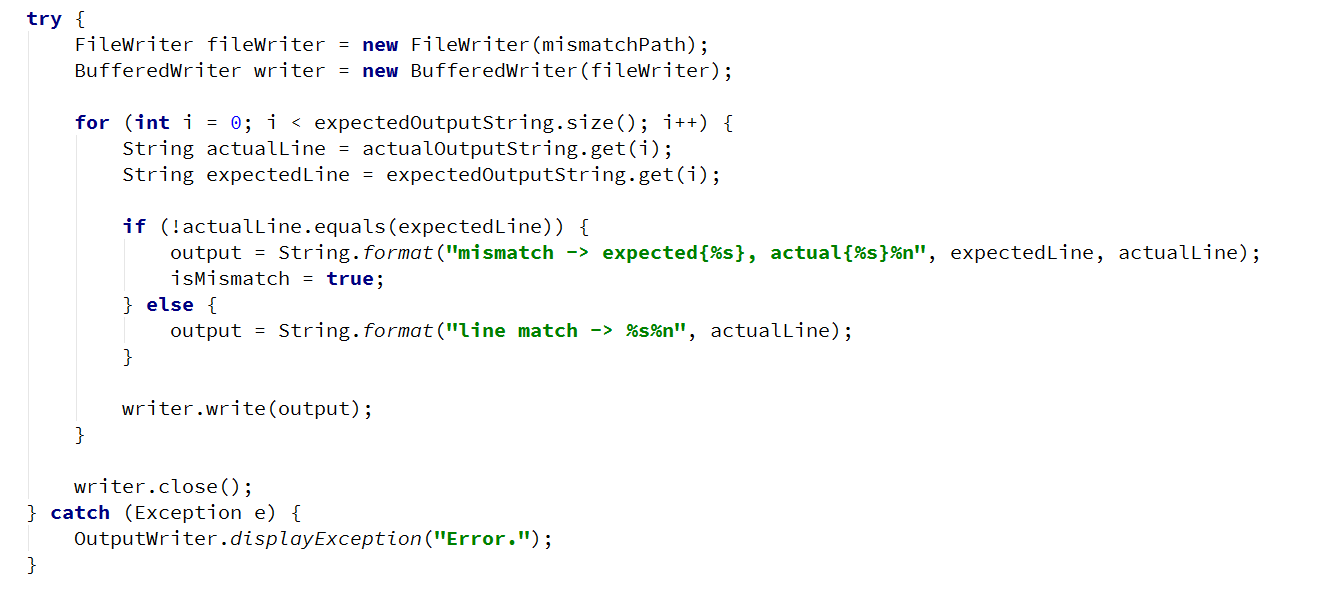


Try testing again. You should now get a proper result:

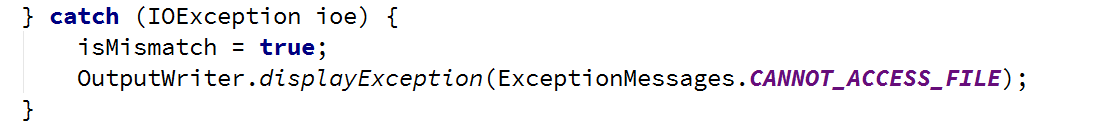


## Comparing Files in the Tester Class

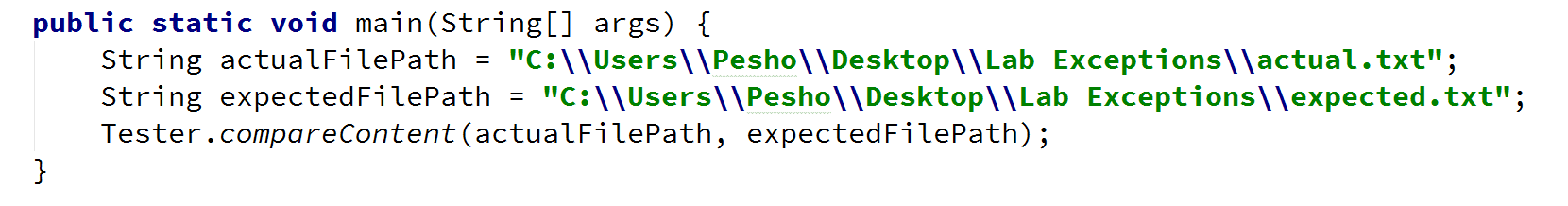
You already tested file comparison in the previous labs but what will happen if one of the files is missing? We are throwing an exception of type Exception that is the ancestor to all exceptions. The problem is that this is not very descriptive and if the program breaks the user will not know what caused the problem. So now we are going to handle the exception properly. This is what we had until now in the compareStrings() method:

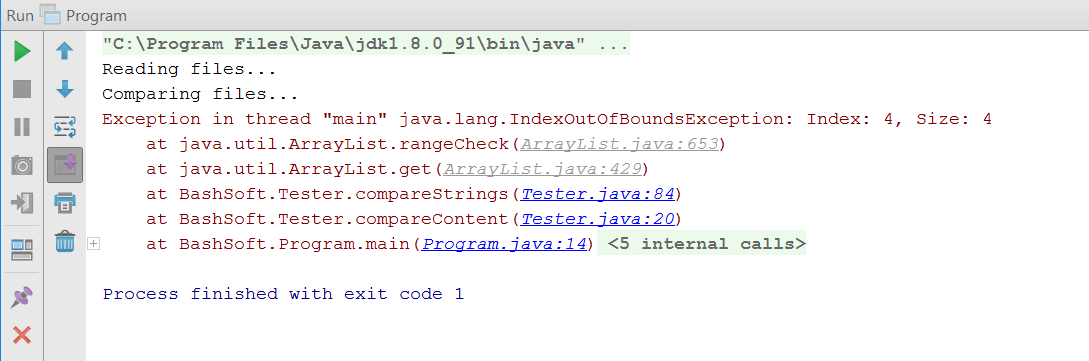


We need to change the type from Exception to IOException and we will print an appropriate message. For this we create a final message in the ExceptionMessages class CANNOT\_ACCESS\_FILE with a message “Cannot access file” and we should get something like this. Also we need to set isMismatch to true:



But we still have a problem. Imagine that the files that we are comparing are with different length. In fact, try comparing the files provided with this lab exercise (change file paths accordingly) and you should get a nasty IndexOutOfBoundsException:

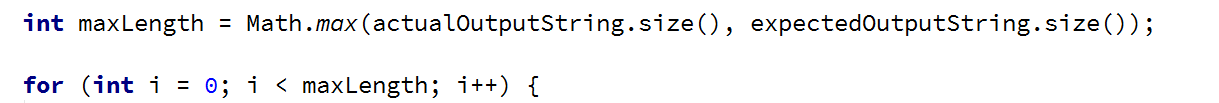




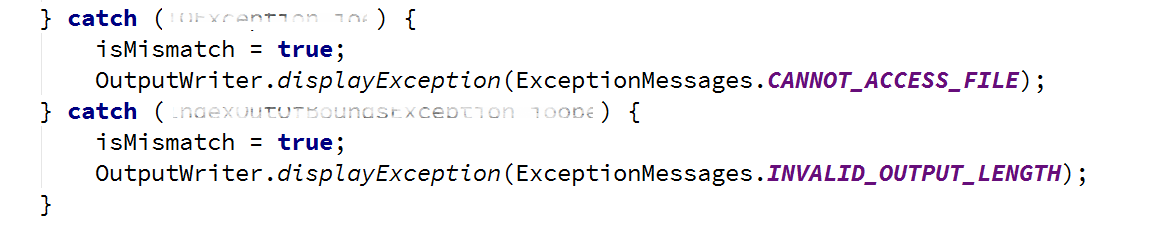
Well you guessed it, we need to handle that as well. First we are going to change the loop bounds. Previously we looped through files until we reached the expected output file size:



Now we are going to loop until we reach the size of the bigger of both files by initializing a helper variable.



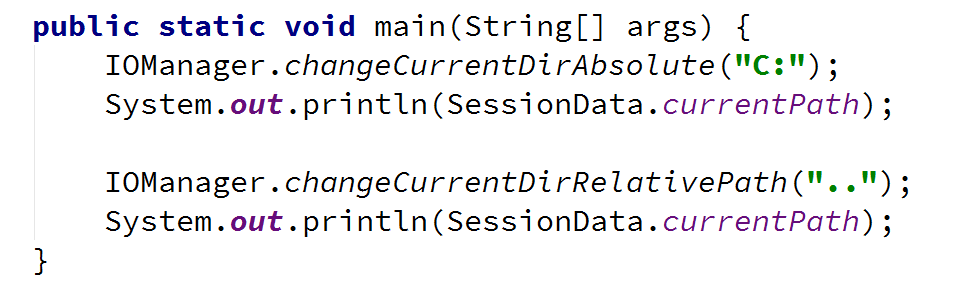
Finally, we can just add one more catch clause to our try-catch block and display some descriptive message. We can call the message INVALID\_OUTPUT\_LENGTH with a string to display “Actual and expected files are with different length.” We pass that to the output writer and set isMismatch to true:



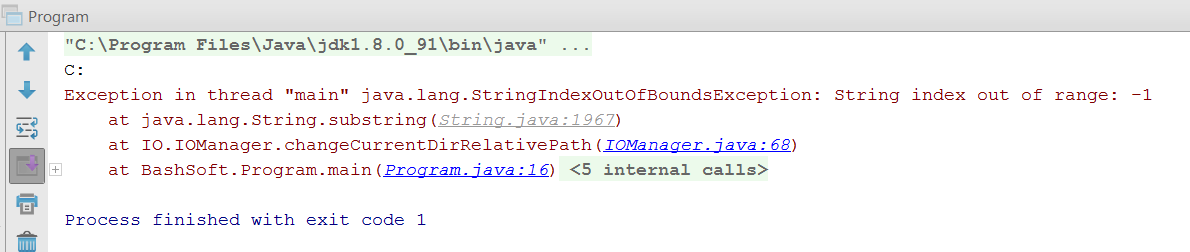
## Going One Folder Up in the Hierarchy

As we know, the logic for the changing of the folders works correctly, but have you tried to go one folder up when you are in the root folder of the partition.

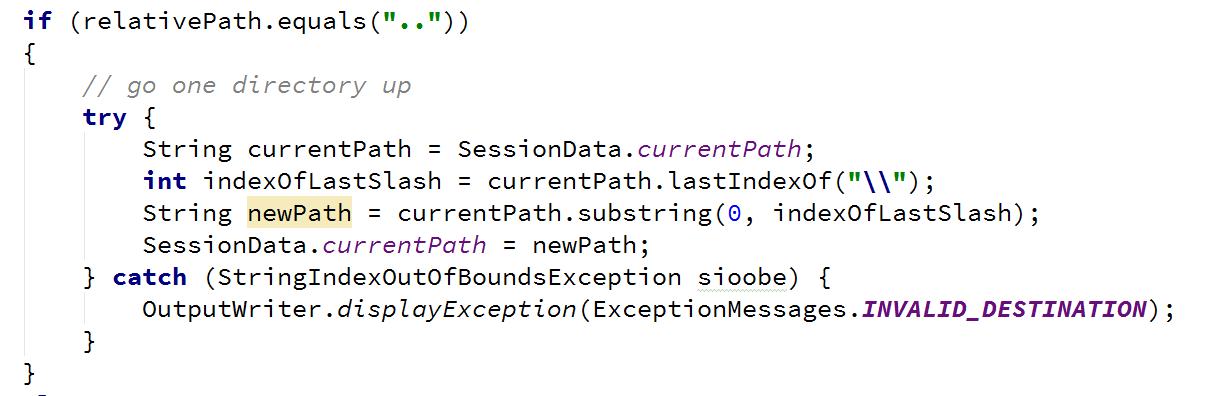
Let’s call the changeCurrentDirectoryRelative with the parameter “..” when our current folder is “C:”, so that we go up one folder above the root.



Yeah, you guessed it. You get an exception because the index of last slash returns -1 and then you are trying to get a substring between 0 and -1:



If we put all the operations that are in the body of the if that checks for the two dots, in a try block, we’ll be able to catch the raised exception in the exact time and print the corresponding message for such a situation. You can create a message INVALID\_DESTINATION with a text “Cannot go higher in folder hierarchy.”:



Congratulations! You’ve successfully completed the Exception Handling Lab.