Exception Handling

Sean Parrott-Wolfe

Just as there are numerous types of exceptions in C#, there are also numerous methods of handling exceptions these exceptions. The standard method for handling simple exceptions has historically been the use of try-catch-finally verbiage in order to keep one’s from application crashing. In such an instance, the program attempts to execute the try block. If errors occur, code under the catch block is executed. The finally block allows for code to be executed regardless of the outcome of the test. For example:

try

{

int num = int.Parse(Console.Readline());

}

catch

{

MessageBox.Show(“NOT an integer!”);

}

finally

{

MessageBox.Show(“Goodbye.”);

}

In the example, the code attempts to get an integer out of the user input. If a non-integer is entered, the catch block executes, displaying the message. Regardless of whether or not an error is thrown, the “Goodbye.” message is displayed. This method of handling exceptions is useful when errors are to be logged, such as in backend environments.

The alternate method for parsing a string is TryParse. In a TryParse block, no catch is necessary, and if the parsing of the text string fails, the program does not throw an exception. For example:

if(int.TryParse(num.Text,out result))

{

MessageBox(“Your number is ” + result);

}

else

{

MessageBox.Show(“NOT an integer!”);

}

In this example, no exception actually occurs, as the failure is an expected possibility. If speed is the goal, or if details about the error are unwanted, this is the preferred method of handling exceptions.

Sources:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/exception-handling>

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/exceptions/>

<https://www.w3schools.com/cs/cs_exceptions.asp>