The Learning House

427 S. 4th Street #300

Louisville KY 40202

Exceptions

.NET Cohort

Coding Bootcamp

SOFTWARE GUILD

Lesson Goals

- Learn what exceptions are
- Learn how to capture exceptions to prevent program crashes
- Learn how the finally block can continue important executions
- Learn how to create your own custom exceptions

What is an Exception?

- An *exception* is a runtime error in a program that violates a constraint or is an unexpected condition during normal operation.
- Good examples are dividing by zero or trying to write to a read-only file.
- Exceptions are *raised* or *thrown* when they occur.

Notice…

- We used the word “unexpected” to define an exception.
- A common coding anti-pattern is for developers to use exceptions to control flow of the application. This is frowned upon.
- Exceptions are for exceptional circumstances. For bad user input and similar problems, gracefully handle them and display a message instead of throwing an exception.

The *try* Statement

- The *try* statement allows us to wrap blocks of code where we think exceptions could occur in order to guard the program from crashing.
- The *catch* statement can contain one or more clauses to handle specific kinds of exceptions.
- The *finally* block contains code that must run regardless of whether or not an exception occurs.

Handling Exceptions

If you catch the exception, your program will not crash. Catching exceptions gives us a chance to restart a process, inform the user, write to a log, and otherwise put the program back into a valid state.

Notice that most of the applications you use don't just close if something small goes wrong.

```
static void HandleException()
{
    try
    {
        int x = 5;
        int y = 0;

        Console.WriteLine(x / y);
    }
    catch
    {
        Console.WriteLine("You did something bad, " +
                          "but I'm going to keep running!");
    }

}
```

The Exception Class

- There are many types of exceptions (like Divide By Zero) that can occur. .NET has many of them built in.
- When exceptions occur, the CLR does the following:
  - Creates an exception object
  - Looks for a catch clause to handle it
  - If it can't find a catch clause, it crashes the program

BCL Exceptions

- The base class library has a few classes available for exception handling:
  - System.Exception: The base class for all exceptions
    - SystemException: The base class for all predefined system exceptions (IndexOutOfRange, NullReference, etc.)
    - ApplicationException: The base class for non-fatal, application-defined exceptions

Important Exception Properties

Property	Type	Description
Message	string	Contains the error message explaining the cause of the exception
StackTrace	string	Contains information describing the call chain of where the exception occurred
InnerException	Exception	If the current exception was raised by another exception, this contains a reference to the parent
HelpLink	string	Allows developers to provide a URL to help information about the cause of the exception
Source	string	By default, the name of the assembly that originated the exception; can be set to other things

3 Ways to Catch

- catch { }
  - Matches any exception
- catch(ExceptionType) { }
  - Only catches exceptions that match the type
- catch(ExceptionType variable) { }
  - Catches the exception of the type, and allows you to manipulate the properties using the variable

Three ways to catch

DEMO

The *Finally* Block

- If an exception occurs in a *try* block, control immediately jumps to the *catch* block, then to the *finally* block.
- If no exception occurs in the *try* block, the *catch* is skipped but *finally* will still execute.

Flowing through the call stack

DEMO

Er… What?

1. CallStackExample() calls Method1(), which calls Method2(), which encounters a DivideByZero Exception.
2. System checks method2 for a catch that matches DivideByZero and doesn't find one, so it executes the finally.
3. System checks method1 for a catch that matches and doesn't find one, so it executes the finally
4. System checks CallStackExample for a catch that matches and finds it, so it executes the catch, then the finally, and then continues execution.

Throwing Exceptions

- You can raise your own exceptions by using the *throw* statement.
- You can also create your own specific exception types by inheriting from the Exception base classes, though this is not common.

Throwing an argument exception

DEMO

Conclusion

- We can catch exceptions and handle them to prevent the application from crashing.
- We shouldn't throw exceptions to control application flow; instead, throwing exceptions should be exceptional.
  - We don't throw exceptions for input validation failures or other things that will "commonly" go wrong. Exceptions are for big problems.