



Module 1 Day 18

File I/O: Writing Files

What makes an application?

- Program Data

- ✓ Variables & .NET Data Types
- ✓ Arrays
- ✓ More Collections (list, dictionary, stack, queue)
- ✓ Classes and objects (OOP)

- Program Logic

- ✓ Statements and expressions
- ✓ Conditional logic (if)
- ✓ Repeating logic (for, foreach, do, while)
- ✓ Methods (functions / procedures)
- ✓ Classes and objects (OOP)
- ❑ Frameworks (MVC)

- Input / Output

- User

- ✓ Console read / write
- ❑ HTML / CSS
- ❑ Front-end frameworks (HTML / CSS / JavaScript)

- Storage

- ❖ File I/O
- ❑ Relational database
- ❑ APIs

Exceptions

- Exceptions are how the .Net Framework reports runtime errors
- Exceptions are thrown when an error occurs
- Your code can catch an error and handle it
 - You can re-throw it using `throw;`
- Examples of runtime errors:
 - Attempting to `int.Parse` a non-numeric value
 - Attempting to read a File that does not exist
 - Divide by zero
 - NULL reference exception
- You can define and throw your own Exceptions
- Exceptions “climb the stack” until someone catches it

Exceptions

```
try
{
    // Do some work here...
}
catch (ArgumentNullException e)
{
    // catch most specific Exceptions first
}
catch (Exception e)
{
    // (optional) catch more general exceptions later
    // (optional) re-throw the same exception so it can be caught further up the stack
    throw;
}
finally
{
    // (optional) Do work that should execute whether the above succeeded or failed
}
```

Let's
Code

Writing to a File

- Use a [StreamWriter](#)
- Write and WriteLine methods
- Flush method writes any *buffered* data
- Dispose does that for you (using)

```
using (StreamWriter sw = new StreamWriter(outPath, false)) // False is default (do not append - overwrite)
{
    sw.Write("Write a portion of a line.");
    sw.WriteLine("Write a line with line-feed.");
} // End of using - buffer is flushed and file is closed
```

Let's
Code