Revature

# Day\_4 File I/O

## Big Picture

* Files save data as a string
* C# Understands C# syntax while your data source has their own way of representing their data
  + There has to be a way for the two languages to communicate between the two.
    - Bytes is the solution
* Convert your files into binary(bytes) such that the data source can understand between the two.
* Saved information:
  + Files: string
  + Databases: entries

## What is a byte stream?

* This is used so that they can be read into your files
* Do we have to know the different types of stream?
  + No! C# is very kind to us to abstract this process.

## Character Stream

* Main difference to byte and character string is that;
  + Character stream is limited to streaming char variables
  + Bytes are no limited as it is essentially machine code that can be interpreted anyway.
    - This means you could possibly send photos as well!

## Additional Read

‌<https://docs.microsoft.com/en-us/dotnet/standard/io/>  
<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/file-system/how-to-write-to-a-text-file>  
[Byte vs Char stream](https://stackoverflow.com/questions/3013996/byte-stream-and-character-stream)

# Serialization

## Serialization

* Our case we save or values into a string which we will save to a file
  + This is because files store data as a string
* Once we return the value, we will unpack the string and then return it back into the previous value (like an object), or Deserialization
* Forms of serialization:
  + Custom Binary
    - This is bugged prone
  + XML
  + JSON
    - C# has built in methods for json

# Application Monitoring

## Application Monitoring

* Highlighting certain events that come up during runtime
* Useful for finding points of errors.

## Breakpoints

* Add breakpoints to line of code, during Debugging, the program will pause at your breakpoints.
* During the pause, you can see the state of all your variables/programs/object ETC, to check why you are getting a certain response.
  + This can be even when the code runs with no Runtime Errors
* Step 1: look into values
* Step 2: Step to your next breakpoint
* **Step Over**; a function
  + Skips over method calls to the next physical line of code
* **Step Into** Function
* **Step Out** of Function
  + Can be used to check function and when satisfied, you can leave the current function being implemented.

## Logging

* Events during Runtime
* Errors are example of elements that are logged
* Business logic, transactions, and other custom needs can be logged.
* This is usually specific to the creators needs

## Logging Levels

**These are in order of severity**

* Verbose
  + ANYTHING AND EVERYTHING
* Debug
  + Only accessible in Development. Internal systems
* Information
  + Something Happened
* Warning
  + Things are degraded or endangered.
* Error
  + Functionality is not available. Calling a method that doesn’t exist
* Fatal
  + Very severe events that can lead to aborting the program. “Who killed me!”
  + “My Name is Antonio Montoya, you killed my code, its time to be removed or updated!” – Debugger

## Logging Levels

# Exceptions

* Exceptions can be bad (On Review days), or can be good (for test cases)
* Events that are fatal to the program runtime, They’re not something that can be handled.
* Sometimes its like stack overflow, like an infinite loop
* Common cause of stack overflow, especially with working with models, is when you set the value of a property, not to the value it was intended for, but instead, set it to the object itself.
* If this happens, then I want you to throw the exception, and do this to handle it

## Exception Hierarchy

* System Exception
  + Built into the System
* Application Exception
  + Custom built Exception
* Internal Exceptions
  + Exceptions built into the internal workings

## Throwing Exceptions

* Your compiler will mention specifically what a problem might be
  + Syntax Errors; something didn’t compile right
  + Runtime Errors; something went wrong at runtime, like trying to access an index of an array that is out of bounds.
* You can make custom exceptions for custom errors?
  + An example would be a Restaurant doesn’t exist that’s being search for

## Handling Exceptions

* Putting your code in a try – catch block will signify that the code is error prone.
* Anything in catch tells the code what to do with the error to make it function.
* Finally is for cleanup, IE closing your DB or other processes that must happen regardless of the results.

## Where the logs at, at, at…?

* Files
  + JSON, XML
* DBs
  + Production/Enterprise level for store history of logs
* Console
  + Like a heathen
* Etcetera
  + DO YOUR RESEARCH!

## Tracing

* Different from logging
  + Tracing follows functionality, logging tracks what error was called.
* Includes information on which functions were called leading up to the error.

## Additional Reading

‌[Intro to Debugging](https://www.fourkitchens.com/blog/article/step-step-through-debugging/)  
[Serilogging](https://stackify.com/serilog-tutorial-net-logging/)  
[Logging Levels](https://nblumhardt.com/2014/03/logging-levels-in-a-structured-world/)  
[Intro to logging](https://www.freecodecamp.org/news/you-should-have-better-logging-now-fbab2f667fac/)  
[Logging in .Net](https://michaelscodingspot.com/logging-in-dotnet/)  
[Tracing vs Logging vs Monitoring](https://www.bmc.com/blogs/monitoring-logging-tracing/)