Section 4 Class Notes

**Errors**

Two different ways to handle errors:

|  |  |
| --- | --- |
| **Return Types** | **Errors** |
| Typically legacy type | First Class Citizen of the Language |
| -Check Return Value | .Net handles errors (not return types)\_ |
| -Response Accordingly | Must handle errors |
| -Query Info |  |

OpenFile(string) -> Pointer

If something goes wrong, the return type will tell you that. It will return null, for instance. ‘I cannot do this operation, but I’m not telling you why’. In a return type environment, you are expected to check the return type value and respond accordingly. Query Information.

So you would first catch the Return Value.

You then have to write code to react to an error return value.

If no error code is generated, then query the information to figure out what happened.

Var error = GetLastError()

{

Switch (Error)

{

Access.denied: do something; break;

FileNotFound: do something; break;

}

}

Errors:

First Class citizen of the language. An error type is returned by the language or platform. You don’t have to query in order to know what the error is.

Var file = OpenFile(“”);

If this fails, it will crash your program. We need to be able to handle errors.

Build out errors based on what occurred. For instance, if a database call fails… have your code try it again before sending in an error.

To generate errors in .Net, use ‘throw’.

Throw Expression of type T, where T is an exception, or derives from it. Exception is the base type for all errors.

throw Et

throw new Exception(); //notifies caller an error has occurred

As soon as a throw method executes, it always terminates the method.

Message – gives general message of what happened. Specifies why it went wrong.

Stack Trace – a call stack trace of where you were when error happened

InnerException – you can chain errors together

Type – what type of error occurred. Basically specifies what went wrong.

**Exception Types:**

**Exception** is the base type.

**Argument Exception** is the base exception for when the arguments (parameters to the functions) are wrong. The code is bad, not the user interaction. Allows you to return which argument (parameter) was wrong.

* ArgumentNullException – defacto exception you throw when what is received is null.
* ArgumentOutOfRangeException – exception you throw where data received is outside the bounds allowed.

**Invalid Operation Exception** – you are trying to do something that is not valid at the time. Ex. submitting order to Amazon without having paid for it. So what you are trying to do does not make sense.

Another example would be trying to send something to the database before connecting to the database.

**Not Implemented Exception** – typically when we are working on code and haven’t finished it.

**Format Exception**  - format of data is not valid.

Exception Types are done to tell the coder what went wrong.

If iValidator fails, return validationexception

Try/Catch – guard code. Inside the Try is the ‘unsafe code’. If any exception is raised by any code inside of the try block, the catch block executes.

rethrow