Jake O'Connor
UAT MS539 "Assignment – Add Exception Handling"

Exception handling and exception prevention are important when constructing any form of application, be it game or enterprise app or medical technology. Preventable exceptions should be avoided wherever possible in the release build of your application, as they are very likely to have an undesired outcome for your end users.

The first line of defense against unwanted exceptions in your code is the Try[___]() paradigm, which many object types in .Net provide for you already. The most common of these methods is probably TryParse(), which can be used on many object types to parse a string value into the corresponding strongly-typed result, while returning a boolean value that tells you if the parse was successful or not. Types such as Dictionary also provide a TryGetValue() method which will, like TryParse(), return a boolean value telling you if the get operation succeeded and if so will include the retrieved value as an out parameter. Try[___]() methods are extremely useful, and should be used wherever possible to avoid and properly handle errors cases in your code.

The last line of defense against exceptions is the *try-catch* block, which will intercede when an exception is thrown anywhere inside the statement body of the *try* block. Once the exception bubbles up to the *try-catch* block, the *catch* statement (or statements) will be compared against the type of exception and if a match is found the *catch* block will execute. A *try-catch* block can have multiple *catch* blocks, each that handles a different exception type, in order to properly handle any exception that is thrown. A *finally* block can also be added to the end of a *try-catch* block (a *try-catch-finally*), which will always be called after the rest of the statement regardless of whether an exception is thrown or not. *Finally* blocks are often used to release unmanaged memory and close streams that may have been opened in the *try* block in the case that a thrown exception would prevent the normal disposal and closure routines from happening.

In the end, it's never possible to fully prepare for any error or exception in a program, but Try methods and *try-catch* blocks can be used to handle knowable unknowns in your code. Try methods should be used wherever possible both to increase code stability and code functionality; by removing assumptions about a string's contents you harden your code. *Try-catch* blocks should be avoided if possible, specifically in fully contained code that should already be bulletproof, but are extremely helpful when interfacing with external libraries and .Net functionality that has no guarantees of succeeding as expected, such as attempting to open a file that's been locked by an anti-virus program.