Guideline: Exception Handling

**Why we need exception handling:**

1. Exceptions promote API consistency.

2. Exceptions integrate well with O-O languages.

3. With return-value-based error reporting, error handling code is happen very near to the code that could fail. But Exception can be centrelized.

4. Error handling code is more easily localized.

5. Error codes can be easily ignored, and often are.

6. Exception can carry very rich information describing the cause of the failure.

7. Exceptions allow for unhandled exception handlers. Normally just report the error and let the app abort. but use the unhandled exception handler, we can also do some extra work, like document recovery in MS Office.

8. Exception promote instrumentation. Exceptions are well-defined method-failure model.

**Exception Throwing:**

It is important to design all public methods of a framework to report method failures by throwing an exception.

* DO NOT return error codes.
* DO report execution failure by throwing exceptions.
* CONSIDER terminating the process by calling System.Environment.FailFast instead of throwing an exception, if your code encounters a situation where it is unsafe for further execution. Like GC failure, Stack Overflow, etc.
* DO NOT use exceptions for the normal flow control, if possible.
* CONSIDER the performance implications of throwing exceptions.
* DO document all exceptions thrown by publicly callable members.
* DO NOT have public members that can either throw or not based on some options, like "throwOnError" parameters.
* DO NOT have public members that return exceptions as the return value or an out parameter.
* CONSIDER using exception builder method.
* DO NOT throw exceptions from exception filter blocks.
* AVOID explicitly throwing exceptions from finally blocks. Implicitly thrown exceptions resulting from calling methods that throw are acceptable.

**Choosing the Right Type of Exception to Throw:**

* CONSIDER throwing existing exceptions residing in the System namespaces instead of creating custom exception types.
* DO create and throw custom exceptions if you have an error condition that can be programmatically handled in a different way than any other existing exception.
* DO NOT create and throw new exceptions just to have your team's exception.
* DO throw the most specific(the most derived) exception that makes sense.

**Error Message Design**

* DO provide a rich and meaningful message text targeted at the developer when throwing an exception.
* DO ensure that exception messages are grammatically correct.
* DO ensure that each sentence of the message text ends with a period.
* AVOID question marks and exclamation points in exception messages.
* DO NOT disclose security-sensitive information in exception messages without demanding apporiate permissions.
* CONSIDER localizing the exception messages thrown by your components.

**Exception Handling:**

* DO NOT swallow errors by cauthing nonspecific exceptions, like System.Exception, in framework code.
* AVOID swallowing errors by catching nonspecific exceptions, like System.Exception, in application code.
* DO NOT exclude any special exceptions when catching for thje purpose of transferring exceptions.like check what e is in catch (Exception e) block.
* CONSIDER catching a specific exception when you understand why it was thrown in a give context and can respond to the failure programmatically.
* DO NOT overcatch. Exceptions should often be allowed to propagate up the call stack.
* DO use try-finally and avoid using try-catch for cleanup code.
* DO prefer using an empty throw when catching and rethrowing an exception, to keep the state when the exception was thrown.
* DO NOT handle non-CLS-compliant exceptions uing a parameterless catch block.

**Wrapping Exceptions**

* CONSIDER wrapping specific exceptions thrown from a lower layer in a more appropriate exception, if the lower layer exception does not make sense in the context of the higher layer operation.
* AVOID catching and wrapping nonspecific exceptions.
* DO specify the inner exceptionwhen wrapping exceptions.

**Using Standard Exception Types**

* DO NOT throw System.Exception or System.SystemException
* DO NOT catch System.Exception or System.SystemException in framework code, unless you intend to rethrow.
* DO NOT catch System.Exception or System.SystemException, except in top-level exception handlers.
* DO NOT throw or derive from System.ApplicationException.
* DO throw an InvalidOperationException if the object is an inappropriate state.
* DO throw ArgumentException or one of its subtypes if bad arguments are passed to a member.
* DO set the ParamName property when throwing one of the ArgumentExceptions.
* DO use value for the name of the implicit value parameter of property setters.
* DO NOT allow publicly callable APIs to explicitly or implicityly throw NullReferenceException, AccessViolationException or IndexOutofRangeException.
* DO NOT explicitly throw StackOverflowException
* DO NOT catch StackOverflowException.
* DO NOT explicitly throw OutofMemoryException(only by the CLR infrastructure). But you can catch it:-).
* DO NOT explicitly throw InteropException, COMException and SEHException.
* DO NOT catch SEHException explicitly.
* DO NOT explicitly throw ExecutionEngineException.

**Design Custom Exceptions**

* AVOID deep exception hierarchies.
* DO derive exceptions from System.Exception or one of the other common base exceptions.
* DO end exception class names with the Exception suffix.
* DO make exceptions serializable. To across application domain and remoting boundaries.
* DO provide (at least) the common constructors.
* DO report security-sensitive information through an override of ToString only after demanding an appropriate permission.
* DO store useful security-sensitive information in a private exception state.
* CONSIDER providing exception properties for programmatic access to extra information relevant to the exception.

**Exceptions and Performance**

* DO NOT use error codes because of concerns that exceptions might affect performance negatively.
* CONSIDER the Tester-Doer pattern for members that might throw exceptions in common senerios to avoid performance problems relative to exceptions. Dangerous to multi-threads.
* CONSIDER the Try-Parse pattern for members that might throw exceptions in common scenarios to avoid performance problems relative to exceptions.
* DO use the profix "Try" and bool return type for try methods.
* Do provide an exception-throwing memebers for each memeber using the Try-Parse pattern.