Exception Handling

Before Exceptions

- Return codes
- Global error variables (errno)
- It requires discipline to check these.
- Code can continue running after an error unless you've checked for every error.

Not Checking Return Codes (Example 0)

- Causes program to fail in subtle ways.
- Problems may appear in a totally different part of the application.
- Sloppy.

Checking Return Codes (Example 1)

- There must be an error checking clause for each and every clause that returns an error code or updates a global error variable.
- Linear relationship.

With Exceptions

- Requires less discipline from lazy programmers.
- Code generally cannot continue running after an error.

Using Exceptions (Example 2)

- Error handling clauses cover greater numbers of error generating clauses.

- Constant relationship.

Most Important Rules for Exceptions

- Use a unique exception type for each and every “throw” clause in the program.

- Include relevant information as proper properties on the exception object.

- This makes it possible to programmatically execute error handling and recovery code based on the nature of the error.

Unique Exception Types

- If you have two places in the code which throw the same exception for the same reason, refactor them into a single method.
- Reuse exception types only by inheritance.
- Organize exceptions into hierarchies that are useful for code that handles exceptions.
- The only place that will reuse an existing exception type is the test suite.

Unique Exception w/Relevant Information (Example 3)

- It is distinct from other errors, even similar errors.
- It includes the unknown facet key as a property.
- It includes the unknown facet key in the human-readable message.
- As it indicates a programming error, it extends IllegalArgumentException.
- Low effort - 13 lines of functional code.

Exception Messages

- Most software throws generic exceptions with messages generated deep within application code.

- Generating an exception message within application code is a layering violation.

- Instead:

    1. Override the getMessage/toString (or equivalent) methods.

    2. Use the named constructor idiom to provide a message to the exception base class constructor.

Named Constructor Idiom (Example 4)

- No layering violation with construction of error message.

- Message can be passed to the base class constructor.

- Also satisfies the rule that you DO NOT DO WORK IN CONSTRUCTORS.

Further Advice for Exceptions

- Don’t squash exceptions.
    - If you intentionally want to discard an exception, at least log it (perhaps at a “DEBUG” error level.)
- Never show a stack trace to an end user.
    - Write the stack trace to a log file instead.

Return of Return Codes

- Even in the world of exceptions, things still return values that have to be inspected.

- In particular, read and write procedures do not guarantee that they actually read or wrote the requested number of bytes.

Read Loop (Example 5)

- Loops until it reads the required number of bytes.
- Signals an error if the file ends before it reads the required number of bytes.
- Uses a unique exception type.
- Uses the named constructor idiom.

Cleanup (Example 6)

- Java, Javascript: finally
  - Note the inner try/catch in the finally clause.
- Other languages have constructs superior to finally.
  - C#, Python: using, with
  - C++: RAII
    - Writing exception-safe C++ code is a topic worthy of its own Tech Talk.

Catching Exceptions in C++

- The rule here is: throw by value, catch by reference.
    - Catch by a const reference if possible.
    - This lets the compiler manage the lifetime of the object for you.
    - Allows you to use polymorphic behavior through the exception reference (e.g. the what method.)

Rethrowing Exceptions (Example 7)

- You probably should have used using/with or RAII instead.
- If you must, then simply “throw” or “raise”.
  - This preserves the original stack trace.
- However, this does not work in Java.
  - Exception.fillInStrackTrace() will fix up the stack frames to point at the new “throw” location.
  - You probably want to chain the exception instead. This preserves the old stack trace and provides an additional new stack trace for the new throw clause.

Exception Chaining (Example 8)

try/catch/else

- Necessary so to refine which statements are covered by catch clauses and which variables are in scope.
- Python-only
    - Which is somewhat strange since Python has function scope.
- Must be simulated in other languages.

try/catch/else (Example 9)

Questions?