Swift Error Handling

GUARD that you DO not TRY to DEFER CATCHing what has been THROWn

High level:

If you’ve dealt languages that made use of try/catch and Exceptions before, there is not a steep learning curve for Swift’s “Error Handling”

The language designers seem to have mostly adopted standard exception handling, with some important changes that address some of the pitfalls in other languages.

Meet the keywords

- throw
- do
- catch
- try
- defer
- guard (useful, but not directly related)

THROW

Basics

- Ends execution of the current scope
- Begin error propagation to enclosing scope
- Operand: Must conform to ErrorType protocol

Of Note

- The current scope must be marked with the “throws” keyword
- ErrorType requires get-able properties _domain : String, and _code : Int
    - It’s free for enums!
- enum-s, especially associated value enum-s, seem the be the idiomatic choice for what to throw.

documentation

DO

Basics

- Introduces a new scope
- Can optionally have `catch` clauses [see next slide]
- It is the analogue of other language's "try" keyword
  - Delimits a new scope that might exit early

Of Note

- No overhead!
- Variables introduced inside are not available outside

documentation

CATCH

Basics

- Works like switch’s “case” - Looks for match, among patterns offered
- Can have a “where” clause
- A catch without a pattern

} catch {

  - works like a switch’s “default”
- If you don’t have an exhaustive collection of catch-s (or if the compiler isn’t certain that you do) error propagation will continue upwards

documentation

TRY

Basics

- Mandatory keyword before functions/methods that “throw” errors
- Enclosing scope must be able to handle errors
    - Either by indicating that it itself “throws” or by catching

Of Note

- This is NOT like other language’s try keyword
- Works as a call out to developers
    - It’s always clear what parts of the code can cause change in program control
- If you’re certain that an error will not be thrown, you can suppress the constraints to the enclosing scope by doing `try!`

DEFER

Basics

- Introduces a new scope
- Adds given chunk of code to a secret LIFO queue associated with current scope that will be executed immediately before program control is transferred.
  - No matter how control is transferred (e.g. throw, return)

Of Note

- Not allowed to transfer program control from within block
  - e.g. no “throw”s or “return”s
- Analogue of other language’s “finally”
  - Here’s some code I need to execute no matter what
  - But it’s more flexible! Use anywhere!

documentation

GUARD

Basics

- Like an "if", but with constraint that following else block MUST transfer program control (e.g. via return or throw)

Of Note

- Can be used to make developer intent explicit! [e.g. early exit/return]
- Conditional binding pattern is inverted in that successfully bound variables are for use *outside/after* the guard block
  - `if let fizz = buzz {/* use fizz */}`
  - `vs.`
  - `guard let fizz = buzz else {return}`
  - `/* use fizz */`

documentation

New Patterns

- guard throw
- do defer try catch
  - other languages would say
    - try catch finally

Transitioning from Objective-C style

Interoperability magic means that some/most/all? NSError** style is exposed to swift in the new style!

[code example in repo]

Final Notes

Swift doesn’t have to “unwind the call stack” when you throw an error.

They claim “throw” has a similar footprint to “return”!

documentation

Remember, what in other languages is:

try->catch->finally

in Swift is

do->defer->try->catch