Protocol Extensions

Extensions can be written for protocols
Methods provided in a protocol extension are used as a default implementation and won’t be present if the adopting class has an implementation
Already being adopted to add some methods to protocols in the API (for example, indexOf global function is now available on collections as a method)

Guard statements

Used to check values and perform optional binding (similar to if/let)
If the optional binding fails or the condition is false, the guard block is executed
Guard blocks must return or throw
Bound variables are in scope after the guard statement succeeds (i.e., does not return/throw)
Deferred Statements

Code that is executed at the end of a method invocation
Executes regardless of if the method returns or throws

Error Handling

3 kinds of errors: simple, complex, fatal
Complex errors previously handled with NSError in/out parameters and an appropriate return value (nil / false)
Swift 2.0 adds a throws keyword for func declarations, a throw keyword for throwing an error, and do/try/catch syntax for calling a func that throws
Can throw anything that adopts the ErrorType protocol (NSError adopts this protocol)

Error Handling, cont.

The try! keyword can be used to call a method that throws without do/catch, but will crash if the method throws
Catch statements must be exhaustive
Current SDK methods that take an NSError inout parameter are automatically translated to the new syntax
No finally block, as there is in Java
Type Casting

The `is` keyword is used to test a type.

The `as?` keyword is used to convert a type into another optional type, which can be unwrapped and used.

The `as!` keyword can be used to force type cast; this will cause a crash if the instance being cast is not a member of the type it is being cast to.

Switch statement cases can test for type conformance using the `as` or `is` keywords.

Documentation

“The Swift Programming Language”