Swift: Access Control

And then there were three...

There are three different access-control levels:

- Public
- Interal
- Private

Public

This access control is used to expose types (classes, enums, structs, protocols) and variables to a target/module other than the one it was defined in.

Internal

The is the default access control level, and is implicit in most contexts.

`internal` types and variables are GLOBAL, but only in a given target/module.

Private

This access control is used to restrict types and variables to ONE source file. Think of this access control when you think of something that belongs in an “.m” file, but not a “.h” file

Wait, wait, do I really need to know this?

While it’s possible to get by without using the access control keywords, it’s really not a good idea.

but why?

In Objective-C, you can hide your ugly implementation details in the “.m” implementation file.

Swift without the “private” keyword is just one, enormous implementation file.

Access Control Interactions

Perhaps the least intuitive part of access control is how they interact with each other.

Bottom - Up: PRIVATE

Think of private as an extremely infectious disease. If a function takes a private type as an argument, or returns a private type, it MUST be marked private.

Not marking it private is saying that it's internal, but part of its definition is private, so the whole thing must be marked private.

Failing to do so is a compile time error

Top - Down: PUBLIC

Think of “public” as being sworn to secrecy on all of your internal and private stuff. You can have them, but you can’t share them.

So, public functions may ONLY have publicly-typed arguments and return types.

Public classes may only have public ancestors.

However, public types are free to conform to private/internal protocols. Think of it as having secret powers.

Best of both worlds...? INTERNAL

As against public, internal is infectious

- Internal types will stop things from being able to be public

As against private, internal is sworn to secrecy.

- Internal types/things can make use of private types/variables, but must not share them

Rule of thumb

Public - We don’t have to worry about this yet.

Internal (Implicit) - Anything that would go in a “.h” header file

Private - Use as much as you can get away with

Thankfully, this is all compile-time; so, the compiler usually has something useful to say if you start doing it wrong.

Swift + Objective-C (redux)

“By default, the generated header (_stdibs-Swift.h) contains interfaces for Swift declarations marked with the public modifier. It also contains those marked with the internal modifier if your app target has an Objective-C bridging header. Declarations marked with the private modifier do not appear in the generated header. Private declarations are not exposed to Objective-C unless they are explicitly marked with @IBAction, @IBOutlet, or @objc as well.”

Swift + Objective-C (redux) (cont.)

So, an Objective-C Bridging Header has the effect of moving your Objective-C code into the “internal” context of your swift code.

Otherwise, it would only have access to the public stuff

@IBAction, @IBOutlet and @objc are just dirty cheats...