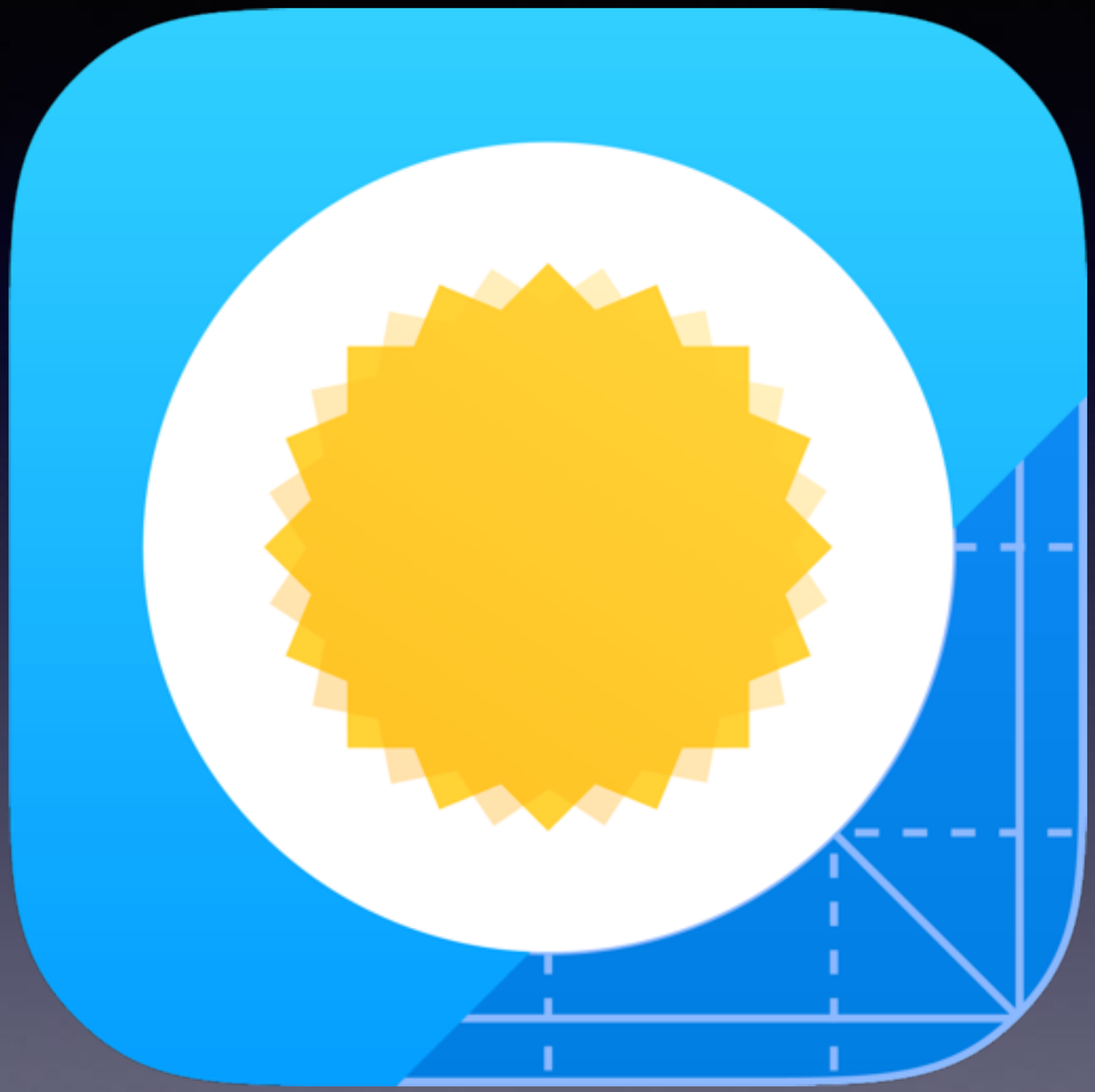


@NSConfArg

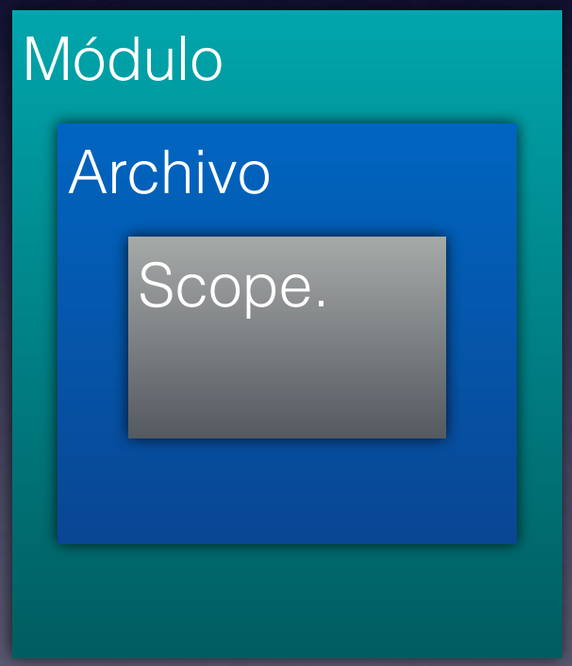
#NSConfArg

Swift, Access Control y Vos

NSConf Argentina 2017

Access Control

- Mecanismo de encapsulamiento.
- Nos permite especificar *quien* puede acceder a *que*.
- Se basa en archivos y módulos.

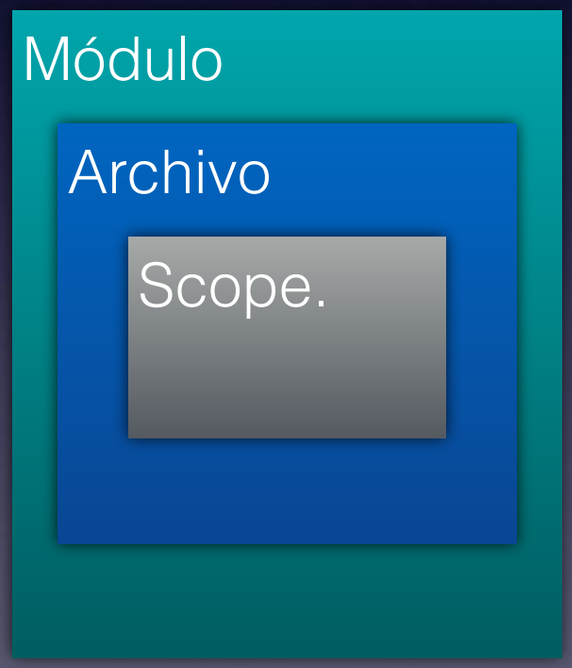


Momento. ¿Módulos?

- La documentación define al “Módulo” como la “Unidad de distribución de código”.
- Los *Packages* del *Swift Package Manager* son módulos.
- Cada *build target* del proyecto es un módulo.
- Los `.framework`s que importamos al proyecto son módulos.

Niveles de Access Control

- `private`: Permite el acceso a una entidad en el *scope* que este se define.
- `fileprivate`: Permite el acceso a una entidad en el *archivo* en el que se define.
- `internal`: Permite el acceso a una entidad dentro del *módulo* en el que se define. Es el default.



Niveles de Access Control

- `public`: Permite el acceso a una entidad fuera del módulo en el que se define.
  Prohibe la herencia y los overrides.

- `open`: Permite el acceso a una entidad fuera del módulo en el que se define.
  Permite la herencia y los overrides.

`public` vs `open`

Cuando estoy **dentro** del módulo, no hay diferencia entre `internal`, `public` ni `open`.

- Para prohibir la herencia dentro del módulo debo usar final.
- No puede combinarse `open` con `final` en una declaración.

Los principios del Acces Control en Swift

“Ninguna entidad puede ser definida en términos de otra entidad que tenga un nivel de acceso menor (más restrictivo).”

– La documentación

Esto implica:

- El nivel de acceso de una tupla es el más restrictivo de los tipos que la componen.
- Todos los valores de un enumerado reciben el mismo nivel de visibilidad que la declaración del enumerado.
- Una función no puede tener un nivel de acceso mayor que sus parámetros o su retorno.

Otra consecuencia

- Cuando escribimos una extension que declara que conformamos a un protocolo, no se indica en ningún lado el nivel de acceso.

- Pero las reglas siguen aplicando. El nivel de acceso de esa declaración es el menor entre el protocolo y la clase.

- Si una clase declarada `internal` tiene métodods `public` para conformar a un protocolo, la visibilidad del conjunto es `internal`.

Entonces…

- ¿PublicClass conforma a PublicProtocol?

```
public protocol PublicProtocol {
    func aMethod()
}

internal protocol InternalProtocol : PublicProtocol {
    func anotherMethod()
}

extension PublicClass : InternalProtocol {
    public func aMethod() {…}
    internal func anotherMethod() {…}
}
```

“Nada es accesible fuera de un módulo a menos que se lo declare como explícitamente accesible.”

– La documentación

Esto implica

- Una entidad nunca puede ser `public` ni `open` sin que lo declaremos explícitamente de esa manera.

- No se pueden agregar elementos a la interfaz de un módulo de manera accidental.

- Las reglas para `private`, `fileprivate` e `internal` son distintas que de las de `public` y `open`.

De un lado…

- Los miembros tienen por defecto el nivel de acceso de la clase que los contiene.

```
private class PrivateClass {
    func somePrivateMethod() {…} // default private
}

fileprivate class FilePrivateClass {
    private func privateMethod() {…}
    func filePrivateMethod() {…} // default file-private
}

class InternalClass { // default internal
    private func privateMethod() {…}
    fileprivate func filePrivateMethod() {…}
    var internalProperty = 0 // default internal
}
```

Del otro lado…

- Los miembros tienen por defecto nivel de acceso `internal`.

```
public class PublicClass {
    var internalProperty = 0 // default internal
    public var publicProperty = 1
}

open class OpenClass {
    var internalProperty = 0 // default internal
    public var publicProperty = 1
    open var openProperty = 2
}
```

Otras consecuencias

- El *default initializer* de un tipo tiene el mismo nivel de acceso que el tipo. Excepto cuando el tipo es público o abierto, en ese caso el inicializador tiene nivel de acceso `internal`.

- Para que sea `public` o sea `open` hay que escribirlo explícitamente.

Detalles inesperados

- El *memberwise initializer* implícito para las estructuras tiene el menor nivel de acceso de todas sus propiedades.

- Pero si todos los niveles de acceso son `public` el inicializador se limita al nivel de acceso `internal`.

Pero… ¿Por qué?

- Todo lo declaremos con nivel de `public` u `open` necesita ser 100% explícito.
- Eso es tan importante que sacrificamos consistencia en el leguaje para tenerlo.
- ¿Que estamos obteniendo a cambio?
- Conseguimos control a la hora de definir nuestra API.

Construyendo APIs.

"Elegir entre `private` y `fileprivate` es cuestión de estilo. Elegir entre `public` y `open` es un acto de diseño"

–Yo

private, fileprivate, internal

- Se usan “puertas adentro” de nuestro módulo.
- Si todo nuestro código vive en un solo módulo, no necesitamos otros.
- Marcar partes de nuestro código como privadas ya es poner un poco de diseño.
- Pero solo estamos trabajando a nivel de clase o de archivo.

public, open

- Para usar estos dos, ya tengo que haber creado un módulo aparte.

- Eso quiere decir que dentro de mi app ya identifiqué una pieza que se puede desacoplar del resto.

- Una unidad de código que puede vivir por separado. Tal vez hasta se pueda reutilizar.

Esconder los detalles

- Recordemos que todo es interno al módulo hasta que nosotros lo agregamos a la API. Si lo incluyo no es un detalle de implementación.

- Al nombrar algo como public o open, estoy diciendo que es parte esencial de mi API.

- Lo que queda **fuera** de la API es igual de importante que lo que queda **adentro**.

Diseñar es difícil.

- La API pública de mi módulo empieza vacía.
- Le agrego partes escribiendo `public` y `open`.
- Cada elemento que agrego a la interfaz es un compromiso con quienes la consumen.
- Las decisiones que hacen a la interfaz son 100% nuestras. Al igual que las consecuencias de esas decisiones.

Módulos != panacea

- Los módulos complican la compilación.
- El diseñar una API es trabajoso y a veces requiere iterar para que salga bien.
- Con un gran poder viene una gran responsabilidad.
- También tengo que pensar en versionado y compatibilidad.

Miren a ambos lados antes de escribir `public`.

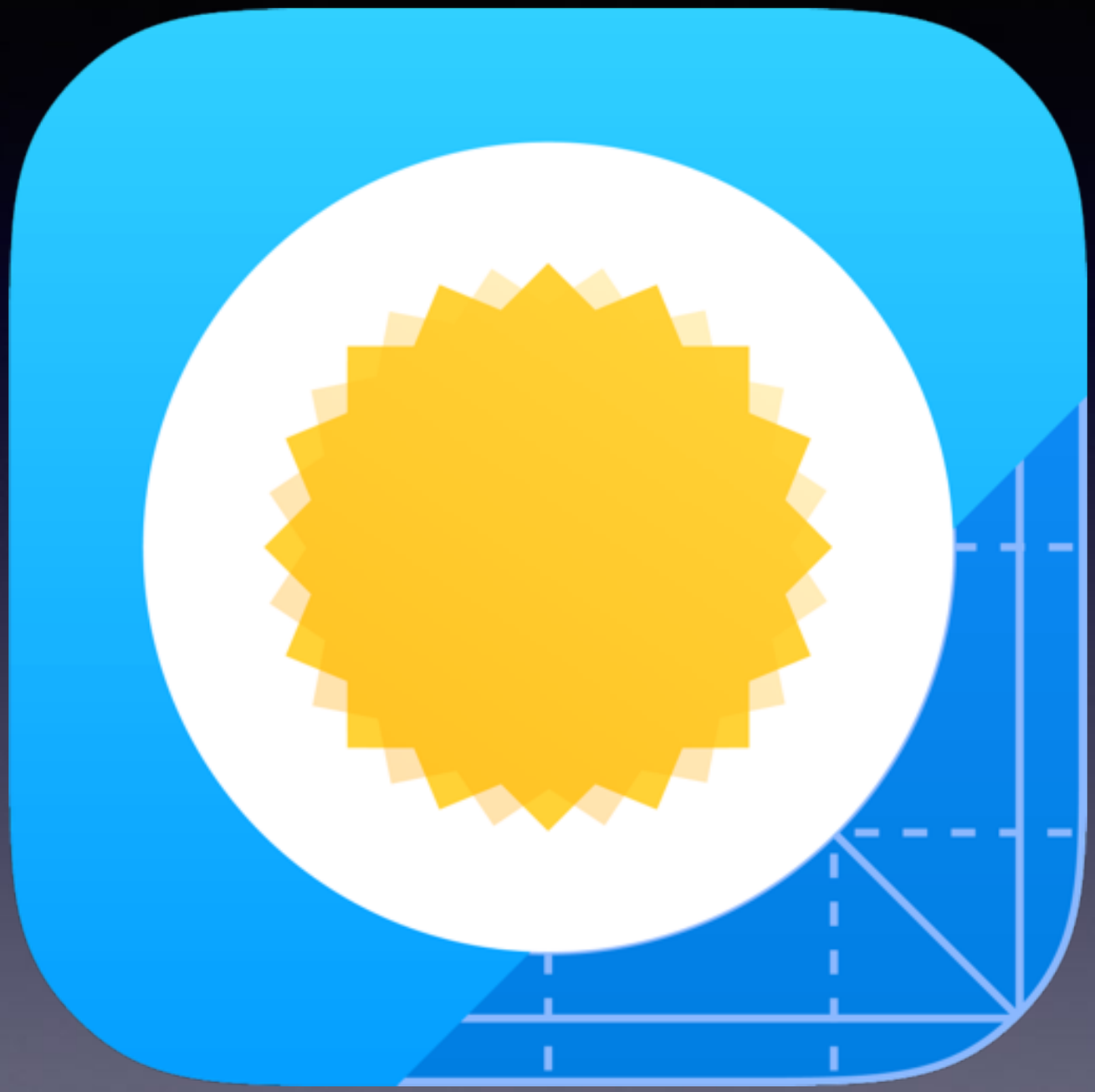
¿Preguntas?

¡Gracias!

¡Muchas gracias!

Los esperamos para la

NSConf Argentina 2018



@NSConfArg

#NSConfArg