Obj-C / Swift Interop

A story of Optionals and Generics

Florent Vilmart
ampme

@flovilmart

Why??

Swift is...

- Modern
- Safe
- Concise
- Readable
- Fast
- Powerful

… Obj-C is

- Old
- Unsafe
- Verbose
- Cluttered
- Fast
- Powerful

MAKE OBJ-C SAFE AGAIN

Imports

Use a Swift class in Obj-C

SWIFT_OBJC_INTERFACE_HEADER_NAME

#import “MyApp-Swift.h”

#import <MyFramework/MyFramework-Swift.h>

Use Obj-C in Swift

SWIFT_OBJC_BRIDGING_HEADER

In MyApp-Bridging-Header.h

#import “MyClass.h”

@objc

```swift
// Swift
@objc(ASwiftObject) // @objc is not required here
class SwiftObject: NSObject {
    let string: String = "SomeString"
}

// Obj-C
ASwiftObject *object = [ASwiftObject new];
```

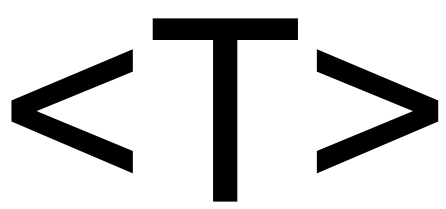
@objc

```
@objc // @objc is required here
protocol MySwiftProtocol {
    func doNothing()
    optional func doNothingElse() // because it's optional
}

#import "CocoaHeads-Swift.h"
@interface Object : NSObject<MySwiftProtocol>
@end

#import "Object.h"
@implementation Object
- (void)doNothing{}
@end
```

The cool kids from Swift land

Optionals

Nullablity



Generics

Lightweight
Generics

Rules of the Swift club

NEVER EVER USE !

When in doubt, refer to rule 1.

If you do...

Optionals

```
@interface AnObject: NSObject

@property (nonatomic, copy) NSString* string;

@end
```

```
// Swift equivalent
class AnObject: NSObject {
    var string: String!
}
```

```
let object = AnObject()
var objectString = object.string // String!
objectString.appendContentsOf("SwiftObject")
// fatal error: unexpectedly found nil while unwrapping an Optional value
```

NS_ASSUME_NON_NULL_*

```
// Obj-C
NS_ASSUME_NONNULL_BEGIN
@interface AnObject: NSObject

@property (nonatomic, retain) AnObject* child;
- (NSString *)aString; // return @”Hello World”

@end
NS_ASSUME_NONNULL_END
```

NS_ASSUME_NON_NULL_*

```
// Swift
let object = AnObject()
assert(object.aString() == "Hello World")
let child = object.child // AnObject
child.aString() // ??
```

A weird case

```
// Obj-C
NS_ASSUME_NONNULL_BEGIN
@interface AnObject: NSObject

@property NSString* string;

@end
NS_ASSUME_NONNULL_END

// Swift
var string = object.string
assert(string == "")
string.appendContentsOf("anotherString")
assert(string == "anotherString")
```

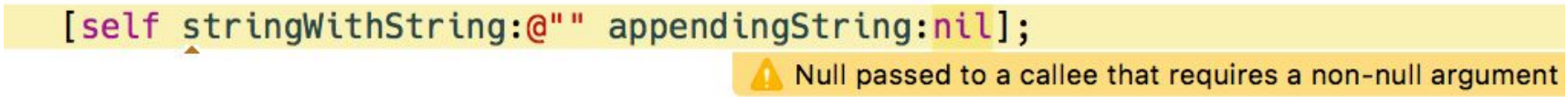
Obj-C and nullability

Operator	Obj-C	Swift
nullable	`@property (nullable) NSString* string;`	`var string: String?`
_Nullable	`NSString* _Nullable string;`	`var string: String?`
nonnull	`@property (nonnull) NSString* string;`	`var string: String`
_Nonnull	`NSString* _Nonnull string;`	`var string: String`

Obj-C and nullability

Ex:

+ (NSString* _Nonnull)stringWithString:(NSString* _Nonnull) string appendingString:(nonnull NSString*) otherString;



Generics

```
// Obj-C
NSArray* generate() {
    return @[@"1", @"String"];
}

// Swift
let array = generate() // array is [AnyObject!]
```

Generics Interop

```
// Obj-C
NSArray<NSString*>* generate() {
    return @[@"1", @"String"];
}

// Swift
let array = generate()
// array is [String]
```

Lightweight Generics

```
NSArray<NSString *> * myArray = @[@"Hello", @"World"];
NSNumber *obj = [myArray firstObject];
```

⚠ Incompatible pointer types initializing 'NSNumber *' with an expression of type 'NSString * _Nullable'

```
NSMutableArray<NSString *> * myArray = [@[@"Hello", @"World"] mutableCopy];
[myArray insertObject:@(0) atIndex:0];
```

⚠ Incompatible pointer types sending 'NSNumber *' to parameter of type 'NSString * _Nonnull'

```
NSMutableDictionary<NSString *, AnObject *>* dict = [NSMutableDictionary new];
```

Lightweight Generics

```
@interface SomeObject<__covariant ObjectType>: NSObject

@property ObjectType child;
- (id)initWithChild:(ObjectType)child;

@end

@implementation SomeObject
- (id)initWithChild:(id)child {
    // ...
}
@end
```

DOES NOT WORK WITH SWIFT

Conclusion
And
Questions