Swift: New Paradigms for iOS Development

Marc Prud'hommeaux

Indie Software Developer / marc@impathic.com

GOTO Conference

Århus, September 30, 2014

Overview

1. Swift
2. FP
3. Tweet
4. HOF
5. Bork
6. Questions?

Flash-Poll

Who Programs What?

Tweet!

```objc
@interface Tweet : NSObject // Tweet.h
@property (nonatomic, strong) NSString *username;
@property (nonatomic, strong) NSString *body;

- (instancetype)initWithUsername:(NSString *)username  body:(NSString *)body;
@end

@implementation Tweet // Tweet.m
- (instancetype)initWithUsername:(NSString *)username
    body:(NSString *)body {
    if (self = [super init]) { self.username = username; self.body = body; }
    return self;
}
@end

@interface TweeterService : NSObject
- (void)tweet:(Tweet *)tweet;
+ (TweeterService *)sharedService;
@end

TweeterService *service = [TweeterService sharedService];
Tweet *tweet = [[Tweet alloc] initWithUsername:@"gotocph"
    body:@"Come see my talk at GOTO!"];
[service tweet:tweet];
```

OBJ-C

meh

```
struct Tweet {
    var username: String
    let body: String
}

class TweeterService {
    func tweet(tweet: Tweet) { ... }
    class func sharedService() -> TweeterService { ... }
}

let service = TweeterService.sharedService()
let message = "Come see my talk at GOTO!"
var tweet = Tweet(username: "gotocph", body: message)
service.tweet(tweet)
```

Swift & ObjC: The Similarities

- Classes
- Methods
- Protocols (interfaces)
- Extensions (categories)
- Functions (methods)
- Semi-automatic memory management (ARC)
- Closures (blocks)

Swift: Small Additions

- Swift Structures
- Namespaces
- Swift Constants
- Operator Overloading
- ObjC Interop

Swift: Big Additions

- Swift Enumerations
- Optionals (non-nullable properties)
- Generics
- Type Inference
- Immutability Support
- Tuples
- First-class Functions

Swift: Big Additions

- Swift Enumerations
- Optionals (non-nullable properties)
- Generics
- Type Inference
- Immutability Support
- Tuples
- **First-class Functions**

```
struct Tweet {
    var username: String
    let body: String
}

class TweeterService {
    func tweet(tweet: Tweet) { ... }
    class func sharedService() -> TweeterService { ... }
}

let service = TweeterService.sharedService()
let message = "Come see my talk at GOTO!"
var tweet = Tweet(username: "gotocph", body: message)
service.tweet(tweet)
```

Bug: No completion notification

```
class TweeterService {
    func tweet(tweet: Tweet) { ... }
}
```

Bug: No completion notification

```
class TweeterService {
    func tweet(tweet: Tweet) -> Void { ... }
}
```

Change the return type

```
class TweeterService {
    func tweet(tweet: Tweet) -> Bool { ... }
}
```

Buzzzz! Wrong!

```
class TweeterService {
    func tweet(tweet: Tweet) -> Bool { ... }
}
```

The Delegate Pattern (ObjC)

```
@protocol TweetDelegate
- (void)tweet:(Tweet *)tweet completed:(BOOL)successful;
@end

@interface TweeterService : NSObject
@property (weak) id<TweetDelegate> serviceDelegate;

- (void)tweet:(Tweet *)tweet;
+ (TweeterService *)sharedService;
@end

TweeterService *service = [TweeterService sharedService];
id<TweetDelegate> delegate = [[MyTweetDelegate alloc] init];
service.serviceDelegate = delegate;

[service tweet:tweet];
```

The Delegate Pattern (Swift)

```
protocol TweeterDeleate {
    func tweetCompleted(tweet: Tweet, success: Bool)
}

class TweeterService {
    var delegate: TweeterDelegate?

    func tweet(tweet: Tweet) { ... }

    class func sharedService() -> TweeterService { ... }
}

let service = TweeterService.sharedService()
let serviceDelegate = MyTweeterDeleate()
service.delegate = serviceDelegate

service.tweet(tweet)
```


FP

FP

“Functional Programming”

FP

=> First-Class Functions

Functional Programming is a Style

Swift has many features that *enable* programming in the Functional Style

A Functional Language compels Functional Programming

Swift is not really a Functional Language

Swift Closures

```
class TweeterService {
    func tweet(tweet: Tweet, done: (Bool)->Void)
}
```

The Function as a Data Type

```
class TweeterService {
    func tweet(tweet: Tweet, done: (Bool)->Void)
}

var donefun: (Bool)->Void

donefun = { success in
    if success {
        println("Tweet successful!")
    } else {
        println("Tweet failed!")
    }
}

service.tweet(tweet, done: donefun)
```

Defining the Function Inline

```
class TweeterService {
    func tweet(tweet: Tweet, done: (success: Bool)->Void)
}

service.tweet(tweet, done: { (success: Bool) in
    if success {
        println("Tweet successful!")
    } else {
        println("Fail Whale")
    }
})
```

And More Succinctly...

```
class TweeterService {
    func tweet(tweet: Tweet, done: (success: Bool)->Void)
}

service.tweet(tweet) {
    println($0 ? "Tweet successful!" : "Fail Whale")
}
```

HOF

"Higher Order Function"

MAP
FILTER
REDUCE

oh my!

MAP

Transform some Stuff into other Stuff

FILTER

Turn some Stuff into fewer Stuff

REDUCE

Turn some Stuff into a single Thing

A Danish Tweet[2]

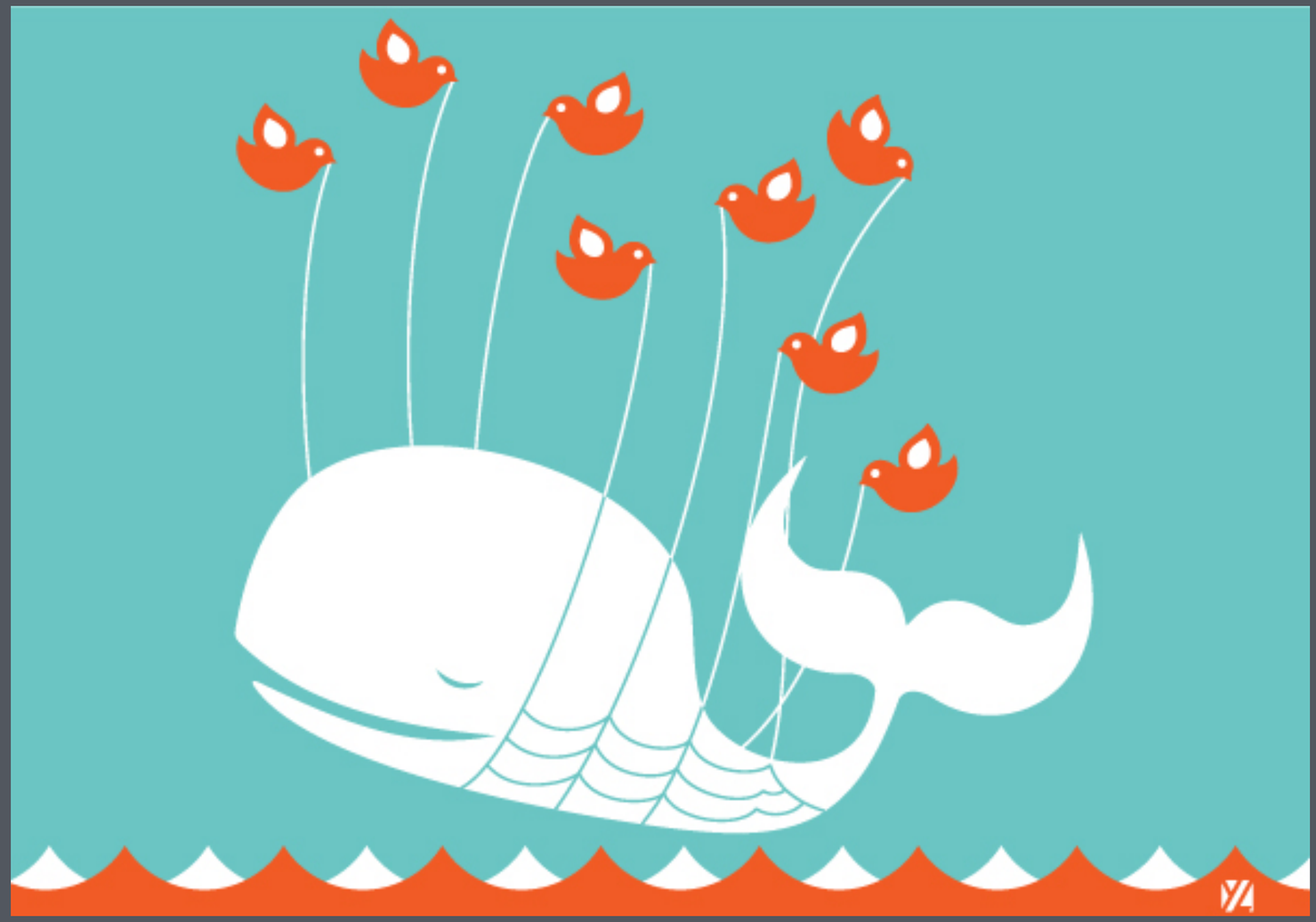
```
let danishTweet = Tweet(username: "Hamlet",
    body: "Tweets, tweets, tweets")

service.tweet(danishTweet, done: { (success) -> Void in
    if success {
        println("Be")
    } else {
        println("Not to be")
    }
})
```

[2] William Shakespeare, Hamlet, Act II, Scene 2 (paraphrased)

```
let adviceTweet = Tweet(username: "Polonius",
    body: "Yet here, Laertes? Aboard, aboard, for shame! "
        + "The wind sits in the shoulder of your sail "
        + "And you are stayed for. There, my blessing with thee. "
        + "And these few precepts in thy memory "
        + "Look thou character. Give thy thoughts no tongue, "
        + "Nor any unproportioned thought his act. "
        + "Be thou familiar but by no means vulgar. "
        + "Those friends thou hast, and their adoption tried, "
        + "Grapple them unto thy soul with hoops of steel, "
        + "But do not dull thy palm with entertainment "
        + "Of each new-hatched, unfledged comrade. Beware "
        + "Of entrance to a quarrel, but being in, "
        + "Bear ’t that th' opposèd may beware of thee. "
        + "Give every man thy ear but few thy voice. "
        + "Take each man’s censure but reserve thy judgment. "
        + "Costly thy habit as thy purse can buy, "
        + "But not expressed in fancy—rich, not gaudy, "
        + "For the apparel oft proclaims the man, "
        + "And they in France of the best rank and station "
        + "Are of a most select and generous chief in that. "
        + "Neither a borrower nor a lender be, "
        + "For loan oft loses both itself and friend, "
        + "And borrowing dulls the edge of husbandry. "
        + "This above all: to thine own self be true, "
        + "And it must follow, as the night the day, "
        + "Thou canst not then be false to any man. "
        + "Farewell. My blessing season this in thee.")
```

Fail!



The Problem Domain

```
let charCount = countElements(tweet.body)
```

== 1,187 characters

countElements() is a global function in Swift

Translating & Compressing Tweets

The Swedish Chef

Börk


Børk

Börk

Børk

HØF

Split

```
let words: [String] = split(tweet.body, { (c: Character) in c == " " })
```

or just:

```
let words = split(tweet.body, { $0 == " " })
```

Words, words, words

```
words = ["Yet", "here", "Laertes?", "Aboard",
"aboard", "for", "shame!", "The", "wind", "sits",
"in", "the", "shoulder", "of", "your", "sail",
"And", "you", "are", "stayed", "for", "There",
"my", "blessing", "with", "thee", "And", "these",
"few", "precepts", "in", "thy", "memory", "Look",
"thou", "character", "Give", "thy", "thoughts",
"no", "tongue", "Nor", "any", "unproportioned",
"thought", "his", "act", "Be", "thou",
"familiar", "but", "by", "no", "means", "vulgar",
"Those", "friends", "thou", "hast", "and",
```

Filter

```
let bigWords = words.filter({ (word: String) in countElements(word) > 7 })
```

More Compactly:

```
let bigWords = words.filter({ countElements($0) > 7 })
```

Map

```
let borks: [String] = bigWords.map({ (word: String) in "Børk" })
```

More Compactly:

```
let borks = bigWords.map({ word in "Børk" })
```

Reduce

```
let translation: String = borks.reduce("",
    combine: { (word1: String, word2: String) -> String in
    return word1 + word2
})
```

More Compactly:

```
let translation = borks.reduce("", combine: { $0 + $1 })
```

Even More Compactly:

```
let translation = borks.reduce("", combine: +)
```

Function Composition

```
let translation = split(tweet.body, { $0 == " " })
    .filter({ countElements($0) > 7 })
    .map({ word in "Børk" })
    .reduce("", combine: +)

let translatedTweet = Tweet(username: tweet.username,
    body: translation)

countElements(translation) // == 125!
```

Børk

Børk Børk

Børk Børk Børk

Børk Børk Børk Børk Børk

Børk Børk Børk Børk Børk Børk Børk Børk Børk Børk Børk Børk Børk Børk

So What?

We're not doing anything here we couldn't do in another other modern language

```
// split the tweet into words
NSMutableArray *words = [NSMutableArray array];
NSMutableString *currentWord = [NSMutableString string];
for (int i = 0, ii = tweet.body.length; i < ii; i++) {
    unichar currentChar = [tweet.body characterAtIndex:i];
    if (currentChar == ' ') {
        [words addObject:currentWord];
        currentWord = [NSMutableString string];
    } else {
        [currentWord appendFormat:@"%c", currentChar];
    }
}

// filter out the shorter words
for (int j = words.count - 1; j >= 0; j--) {
    NSString *currentWord = words[j];
    if (currentWord.length < 8) {
        [words removeObjectAtIndex:j];
    }
}

// map each element of the words array to Børk
for (int k = 0, kk = words.count; k < kk; k++) {
    words[k] = @"Børk";
}

// reduce the words back into a new tweet
NSMutableString *translatedTweet = [NSMutableString string];
for (int l = 0, ll = words.count; l < ll; l++) {
    [translatedTweet appendString:words[l]];
}
```

```objc
// split the tweet into words
NSMutableArray *words = [[tweet.body componentsSeparatedByString:@" "]
    mutableCopy];

// filter out the shorter words
[words filterUsingPredicate:[NSPredicate predicateWithBlock:
    ^BOOL(id evaluatedObject, NSDictionary *bindings) {
    return [evaluatedObject length] > 7;
}]];

// map each element of the words array to Børk
for (int k = 0, kk = words.count; k < kk; k++) {
    words[k] = @"Børk";
}

// reduce the words back into a new tweet
NSString *translatedTweet = [words componentsJoinedByString:@""];
```

```
let translation = split(tweet.body, { $0 == " " })
    .filter({ countElements($0) > 7 })
    .map({ word in "Børk" })
    .reduce("", combine: +)
```

Write Less Code

Write Less Code

< === >

Write Less Code

< === >

(less is more)

FP Checklist

Goal: Reduce Complexity

1. Stop branching (use closures)
2. Stop looping (use closures)
3. Reduce state (use immutability)

Swift's Dark Underbelly

1. It's new, and somewhat buggy
2. Cumbersome bridging
3. No more dynamic dispatch
4. Duck-typing is no longer fun
5. No error-handling

Thank You!

Please evaluate this talk with the GOTO Mobile App

Marc Prud'hommeaux / marc@impathic.com

but you can't follow me on Twitter