Protocols

Matthias Tretter, @myell0w

Protocols

- Similar to `@interface`, but someone else does the implementing

```
@protocol Foo <Other, NSObject> // implementors must implement Other and NSObject too
- (void)doSomething;    // implementors must implement this (methods are @required by default)
@optional
- (int)getSomething;                    // implementors do not have to implement this
- (void)doSomethingOptionalWithArgument:(NSString *)argument; // also optional
@required
- (NSArray *)getManySomethings:(int)howMany;      // back to being "must implement"
@property (nonatomic, strong) NSString *fooProp;  // note that you must specify strength
                                                  // (unless it's readonly, of course)
@end
```

The `NSObject` protocol includes most of the methods implemented by `NSObject`.
Many protocols require whoever implements them to basically "be an `NSObject`"
by requiring the `NSObject` protocol as a "sub-protocol" using this syntax.

Protocols

- Similar to `@interface`, but someone else does the implementing

```
@protocol Foo <Other, NSObject>  // implementors must implement Other and NSObject too
- (void)doSomething;    // implementors must implement this (methods are @required by default)
@optional
- (int)getSomething;                    // implementors do not have to implement this
- (void)doSomethingOptionalWithArgument:(NSString *)argument; // also optional
@required
- (NSArray *)getManySomethings:(int)howMany;       // back to being "must implement"
@property (nonatomic, strong) NSString *fooProp;   // note that you must specify strength
                                                   // (unless it's readonly, of course)
@end
```

- Protocols are defined in a header file

  Either its own header file (e.g. `Foo.h`)

  Or the header file of the class which wants other classes to implement it

  For example, the `UIScrollViewDelegate` protocol is defined in `UIScrollView.h`

Protocols

- Classes then say in their `@interface` if they implement a protocol

```
#import “Foo.h”                    // importing the header file that declares the Foo @protocol
@interface MyClass : NSObject <Foo> // MyClass is saying it implements the Foo @protocol
    …
@end
```

- You must implement all non-`@optional` methods

  Or face the wrath of the compiler if you do not.

- We can then declare `id` variables with a protocol requirement

```
id <Foo> obj = [[MyClass alloc] init];  // compiler will love this!
id <Foo> obj = [NSArray array];  // compiler will not like this one bit!
```

- Also can declare arguments to methods to require a protocol

```
- (void)giveMeFooObject:(id <Foo>)anObjectImplementingFoo;
@property (nonatomic, weak) id <Foo> myFooProperty; // properties too!
```

  If you call these and pass an object which does not implement `Foo` … compiler warning!

Protocols

- Just like static typing, this is all just compiler-helping-you stuff
  It makes no difference at runtime
- Think of it as documentation for your method interfaces
  It's another powerful way to leverage the id type
- #1 use of protocols in iOS: delegates and data sources
  A delegate or dataSource is pretty much always defined as a weak @property, by the way.
  `@property (nonatomic, weak) id <UISomeObjectDelegate> delegate;`
  This assumes that the object serving as delegate will outlive the object doing the delegating.
  Especially true in the case where the delegator is a View object (e.g. UIScrollView)
  & the delegate is that View's Controller.
  Controllers always create and clean up their View objects (because they are their "minions").
  Thus the Controller will always outlive its View objects.
  dataSource is just like a delegate, but, as the name implies, we're delegating provision of data.
  Views commonly have a dataSource because Views cannot own their data!

Protocols

```
@protocol UIScrollViewDelegate
@optional
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender;
- (void)scrollViewDidEndDragging:(UIScrollView *)sender willDecelerate:(BOOL)decelerate;
…
@end

@interface UIScrollView : UIView
@property (nonatomic, weak) id <UIScrollViewDelegate> delegate;
@end

@interface MyViewController : UIViewController <UIScrollViewDelegate>
@property (nonatomic, weak) IBOutlet UIScrollView *scrollView;
@end

@implementation MyViewController
- (void)setScrollView:(UIScrollView *)scrollView {
    _scrollView = scrollView;
    self.scrollView.delegate = self;  // compiler won't complain
}
- (UIView *)viewForZoomingInScrollView:(UIScrollView *)sender { return … };
@end
```