The Feel of Objective-C

Jens Peter Hedegård
Trifork
JPH@trifork.com

Credits and Thanks to:
- Kees van Koetsveld, Trifork, CTO
Objective-C

• Start with C
• Add the Smalltalk object model as a library
• Add a little syntax for
  • Class and method definition
  • Method calls
  • A few object literals
Objective-C
Objective-C

• A mashup of two languages
• Smalltalk grafted onto C
• The boundaries are obvious:
  • Non-C-like syntax in special “zones”
  • Flag characters to mark Objective-C zones
  • In C code, objects are opaque
Objective-C: The Language
Calling Methods

Brackets indicate Objective-C call

Java equivalent: `netService.stop()`

Variable containing target object

Message selector
Methods With Arguments

[serviceNameField setEnabled:YES]

[in_stream read:readBuffer maxLength:4096]

(Yes, that method name is “read:maxLength:”)
Interfaces

@interface Album : MusicObject
{
    NSMutableArray *sampleURLs, *sampleTitles;
}

+ (Album*) albumWithEntryID: (NSString*)entryID;
- (PSEntry*) entry;

@property (copy) NSString* entryID;
@end
NSWhat?

- Objective-C has no namespaces
- Libraries (and apps) use prefixes instead
- Many type names begin with “NS” — for NeXTStep
Implementations

// Album.m

@implementation Album

// method definitions go here

@end
Types

- Object variables are usually pointers
  - e.g., NSString *
- Methods can return any C type
  - including object pointers
- use Objective-C method call anywhere
  - an expression is valid
- Parameters can also be any C type
Basic Types

• NSNumber, NSInteger
• NSString
  • special literal syntax: @"foo"
• NSMutableString
• NSArray and NSMutableArray
• NSDictionary and NSMutableDictionary
Allocation

[NSAlert alloc] Allocates uninitialized object
[new_object init] Performs default initialization
[[NSAlert alloc] init] Standard init pattern
[NSAlert new] Rarely used equivalent

NSAlert *alertSheet;
alertSheet = [[NSAlert alloc] init];
Initialization

```objc
[[NSString alloc] init ]
[[NSString alloc] initWithString: username]
[[NSString alloc] initWithFormat:@"%@/%@", parentAbsPath, relativePath]
[[NSString alloc] initWithBytes:value length:strlen(value)]
[[NSString alloc] initWithBytes:value length:strlen(value) encoding:NSASCIIStringEncoding]
[[NSString alloc] initWithData: data encoding: NSUTF8StringEncoding]
[[NSString alloc] initWithContentsOfFile: path]
```
Special values

• self
• super
• nil
Memory Management

- Objective-C v4 supports garbage collection
  - (but not on the iPhone, yet...)
- Manual reference counting
  - `[obj retain]`
  - `[obj release]`
Autorelease Example

// At the beginning of a block, do this:
NSAutoreleasePool* pool=[[NSAutoreleasePool alloc] init];

// Then, within the block and also in methods
// *called* from that block, do things like this:
return [[time retain] autorelease];

// Then, at the end of the block, release the pool:
[pool release];

• There is always an autorelease pool available.

• Allows simpler division of memory management responsibility.
Incremental Typing

• Usually, Objective-C is statically typed
  • (or as static as C will allow)
• The typedef `id` represents “any Objective-C object”
• You can write methods that work on any type
Protocols

• In Smalltalk terminology, a ‘protocol’ is a set of methods that may be implemented by many classes.

• In Objective-C, this was formalized to resemble what you may know as an ‘interface’ in Java.
Protocols

@protocol KeyValueAccess
- valueForKey:(NSString*)key;
- setValue:(id)val forKey:(NSString*)key;
@end

id <KeyValueAccess> obj = ...;
[obj setValue:@”Peter” forKey:@”name”];
Protocols

// intersection types
id <InputStream, OutputStream> stream = ...  

// or even...
NSFooBar <KeyValueAccess> foobar = ...;

// In Java, such types can be used to
// declare Class parameter constraints...
Slut