Working Effectively with Objective-C on iPhone OS

Blaine Garst
Wizard of Runtimes

Working Effectively with Objective-C on iOS 4

Blaine Garst
Wizard of Runtimes

Objective-C is the language of Cocoa Touch. Take an in-depth working tour of Objective-C, from properties and memory management, to integrating your existing C and C++ code with Objective-C. Examine design patterns, exception models, and other important considerations. A valuable session to hone your knowledge of the language.

LISP/Scheme
C
Objective-C
C++
Java
C#
Ruby
Python
JavaScript
PHP
Perl
OCaml
Haskell
Erlang
Go

Syntax: Statements, Control Flow, Operators
Exceptions
Closures (ObjC: ***Blocks***)

Primitives: Numbers, Strings

Aggregates: Structures, Objects
Inheritance
Interfaces (ObjC: ***Protocols***)
Accessors

Collections: Lists, Arrays, Sets, Maps

Platform: Memory Management
Input/Output

Territory

- Map and Go With What You Know
  - Introduce Objective-C Terminology for **Common** Concepts
- Introduce Objective-C **Uncommon** Ideas
  - Blocks
  - @properties—let the compiler write your accessors!
  - Categories—add behavior (methods!) to any class
  - Selectors, Delegates, @optional
- Discuss Cocoa Touch **Patterns**
  - Memory Management—Retain, Release, Autorelease
  - Mutability, Class Clusters, "PLists"

Syntax: Statements, Control Flow, Operators
Exceptions
Closures (ObjC: Blocks)

Blocks

iOS 4 Block APIs

```
typedef void (^ALAssetsGroupEnumerationResultsBlock)(ALAsset *result, NSUInteger index, BOOL *stop);
typedef void (^ALAssetsLibraryGroupsEnumerationResultsBlock)(ALAssetsGroup *group, BOOL *stop);
typedef void (^ALAssetsLibraryAssetForURLResultBlock)(ALAsset *asset);
typedef void (^ALAssetsLibraryAccessFailureBlock)(NSError *error);
typedef void (^ALAssetsLibraryWriteImageCompletionBlock)(NSURL *assetURL, NSError *error);
typedef void (^ALAssetsLibraryWriteVideoCompletionBlock)(NSURL *assetURL, NSError *error);
typedef void (^AudioQueueOutputCallbackBlock)(
typedef void (^AudioQueueInputCallbackBlock)(
- (void)exportAsynchronouslyWithCompletionHandler:(void (^)(void))handler;
typedef void (^AVAssetImageGeneratorCompletionHandler)(CMTime requestedTime, CGImageRef image, CMTime actualTime, AVAssetImageGeneratorResult result, NSError *error);
- (void)loadValuesAsynchronouslyForKeys:(NSArray *)keys completionHandler:(void (^)(void))handler;
- (void)captureStillImageAsynchronouslyFromConnection:(AVCaptureConnection *)connection completionHandler:(void (^)(CMSampleBufferRef imageDataSampleBuffer, NSError *error))handler;
- (id)addPeriodicTimeObserverForInterval:(CMTime)interval queue:(dispatch_queue_t)queue usingBlock:(void (^)(CMTime time))block;
- (id)addBoundaryTimeObserverForTimes:(NSArray *)times queue:(dispatch_queue_t)queue usingBlock:(void (^)(void))block;
CF_EXPORT void CFRunLoopPerformBlock(CFRunLoopRef rl, CFTypeRef mode, void (^block)(void)) CF_AVAILABLE(10_6, 4_0);
typedef void (^CMAccelerometerHandler)(CMAccelerometerData *accelerometerData, NSError *error);
typedef void (^CMGyroHandler)(CMGyroData *gyroData, NSError *error);
typedef void (^CMDeviceMotionHandler)(CMDeviceMotion *motion, NSError *error);
    void (^_callEventHandler)(CTCall*);
@property(nonatomic, copy) void (^callEventHandler)(CTCall*);
    void (^_subscriberCellularProviderDidUpdateNotifier)(CTCarrier*);
@property(nonatomic, copy) void (^subscriberCellularProviderDidUpdateNotifier)(CTCarrier*);
typedef void (^EKEventSearchCallback)(EKEvent *event, BOOL *stop);
- (void)enumerateObjectsUsingBlock:(void (^)(id obj, NSUInteger idx, BOOL *stop))block ;
- (void)enumerateObjectsWithOptions:(NSEnumerationOptions)opts usingBlock:(void (^)(id obj, NSUInteger idx, BOOL *stop))block ;
- (void)enumerateObjectsAtIndexes:(NSIndexSet *)s options:(NSEnumerationOptions)opts usingBlock:(void (^)(id obj, NSUInteger idx, BOOL *stop))block ;
- (NSUInteger)indexOfObjectPassingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate ;
- (NSUInteger)indexOfObjectWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate ;
- (NSUInteger)indexOfObjectAtIndexes:(NSIndexSet *)s options:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate ;
- (NSIndexSet *)indexesOfObjectsPassingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate ;
- (NSIndexSet *)indexesOfObjectsWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate ;
- (NSIndexSet *)indexesOfObjectsAtIndexes:(NSIndexSet *)s options:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id obj, NSUInteger idx, BOOL *stop))predicate ;
- (void)enumerateAttributesInRange:(NSRange)enumerationRange options:(NSAttributedStringEnumerationOptions)opts usingBlock:(void (^)(NSDictionary *attrs, NSRange range, BOOL *stop))block ;
- (void)enumerateAttribute:(NSString *)attrName inRange:(NSRange)enumerationRange options:(NSAttributedStringEnumerationOptions)opts usingBlock:(void (^)(id value, NSRange range, BOOL *stop))
- (void)enumerateKeysAndObjectsUsingBlock:(void (^)(id key, id obj, BOOL *stop))block ;
- (void)enumerateKeysAndObjectsWithOptions:(NSEnumerationOptions)opts usingBlock:(void (^)(id key, id obj, BOOL *stop))block ;
- (NSSet *)keysOfEntriesPassingTest:(BOOL (^)(id key, id obj, BOOL *stop))predicate ;
- (NSSet *)keysOfEntriesWithOptions:(NSEnumerationOptions)opts passingTest:(BOOL (^)(id key, id obj, BOOL *stop))predicate ;
+ (NSExpression *)expressionForBlock:(id (^)(id evaluatedObject, NSArray *expressions, NSMutableDictionary *context))block arguments:(NSArray *)arguments ;
- (id (^)(id, NSArray *, NSMutableDictionary *))expressionBlock;
- (NSDirectoryEnumerator *)enumeratorAtURL:(NSURL *)url includingPropertiesForKeys:(NSArray *)keys options:(NSDirectoryEnumerationOptions)mask errorHandler:(BOOL (^)(NSURL *url, NSError *error
- (void)enumerateIndexesUsingBlock:(void (^)(NSUInteger idx, BOOL *stop))block ;
- (void)enumerateIndexesWithOptions:(NSEnumerationOptions)opts usingBlock:(void (^)(NSUInteger idx, BOOL *stop))block ;
- (void)enumerateIndexesInRange:(NSRange)range options:(NSEnumerationOptions)opts usingBlock:(void (^)(NSUInteger idx, BOOL *stop))block ;
```

Over 100 APIs use Blocks!!!

- (UIBackgroundTaskIdentifier)beginBackgroundTaskWithExpirationHandler:(void(^)(void))handler

Blocks

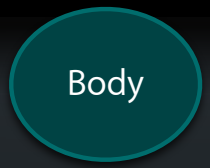
```
C: repeat(10, ^{ putc(d); });

Ruby: z.each {|val| puts(val + d.to_s)}

Smalltalk: 10 timesRepeat:[pen turn:d; draw]

LISP closure: (repeat 10 (lambda (n) (putc d)))

C++0x lambda: template [=](){ putc(d); }
```

Repeat Block *N* Times Function

```
NSMutableString *str = [NSMutableString string];
..
repeat(12, ^{
   [str appendFormat:@"rand: %d ", rand()];
});
```

```
void repeat(int n, void (^blkPtr)(void)) {
    while (n-- > 0) {
        blkPtr();
    }
}
```

Block Literal Syntax Summary




```
^ BOOL (id str)   { return [str length] > num; }

^ int  (int val)  { return rand() % val; }
^      (int val)  { return rand() % val; }   Type inferred!

^ void (id item)  { [item doSomeThings]; }
^      (id item)  { [item doSomeThings]; }   Type inferred!

^      (void)     { [local doSomeThing]; }   Type inferred!
^                 { [local doSomeThing]; }   (void) avoided!
```

About Blocks…

- Start out on stack—fast!
- Blocks are objects!
- Can be copied to heap!
  - [^{...} copy]
  - [block release]
- Used for
  - Enumerations
  - Callback notifications
  - With GCD, moving work off the main thread
  - With GCD, mutual exclusion, concurrency

Introducing Blocks and Grand Central Dispatch on iPhone — Russian Hill, Wednesday 11:30AM

Advanced Objective-C and Garbage Collection Techniques — Pacific Heights, Friday 11:30AM

```
        if ([object isEqual:sought)]) {
            keyForValue = key;
            *stop = YES;
        }
    }];
```

Cocoa Touch Iteration Best Practice

```
for (id e in array) { ..e.. }
```

```
for (id k in dictionary) { ..k.. }
```

Fastest!

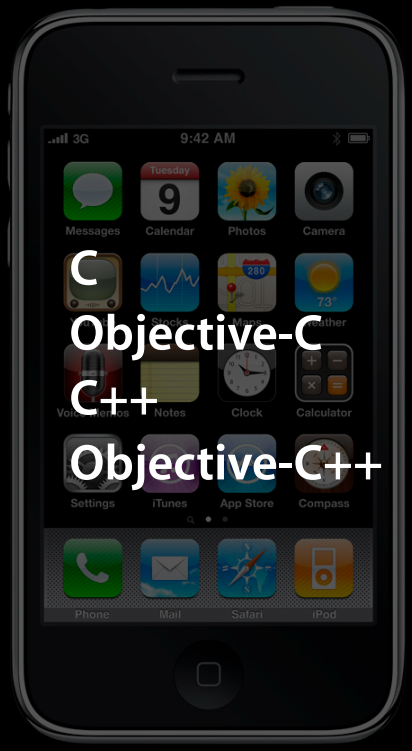
Safest!

Extensible!

For enumerations that need more than one value at a time, use the collection Block APIs:

```
[array enumerateObjectsWithOptions: NSEnumerationReverse
    usingBlock:^(id e, NSUInteger idx, BOOL *stop) { ..e..idx.. }];
```

```
[dict enumerateKeysAndObjectsUsingBlock:
                        ^(id k, id obj, BOOL *stop) { ..k..obj.. }];
```



C

Objective-C

C++

Objective-C++

Objective-C++

MyMixedClass.mm

```
@class MyUIKitWidget;

class MyEngine {
   MyUIKitWidget *widget;
};

@interface MyUIKitWidget : NSObject {
   MyEngine eng;
}
@end

@implementation MyUIKitWidget
- (id)myUIKitWidgetMethod {
   MyEngine ff;
   throw [NSException new];
   return nil;
}
```

- Mix and match:
  - Instance/Member variables
  - Statements
  - Declarations
  - Exceptions
  - Whole Classes
- CAN'T mix methods/functions
- CAN'T subclass one from another

Objects
Inheritance

Terminology for Common Concepts

Table replaces code graphic.

Table look good?

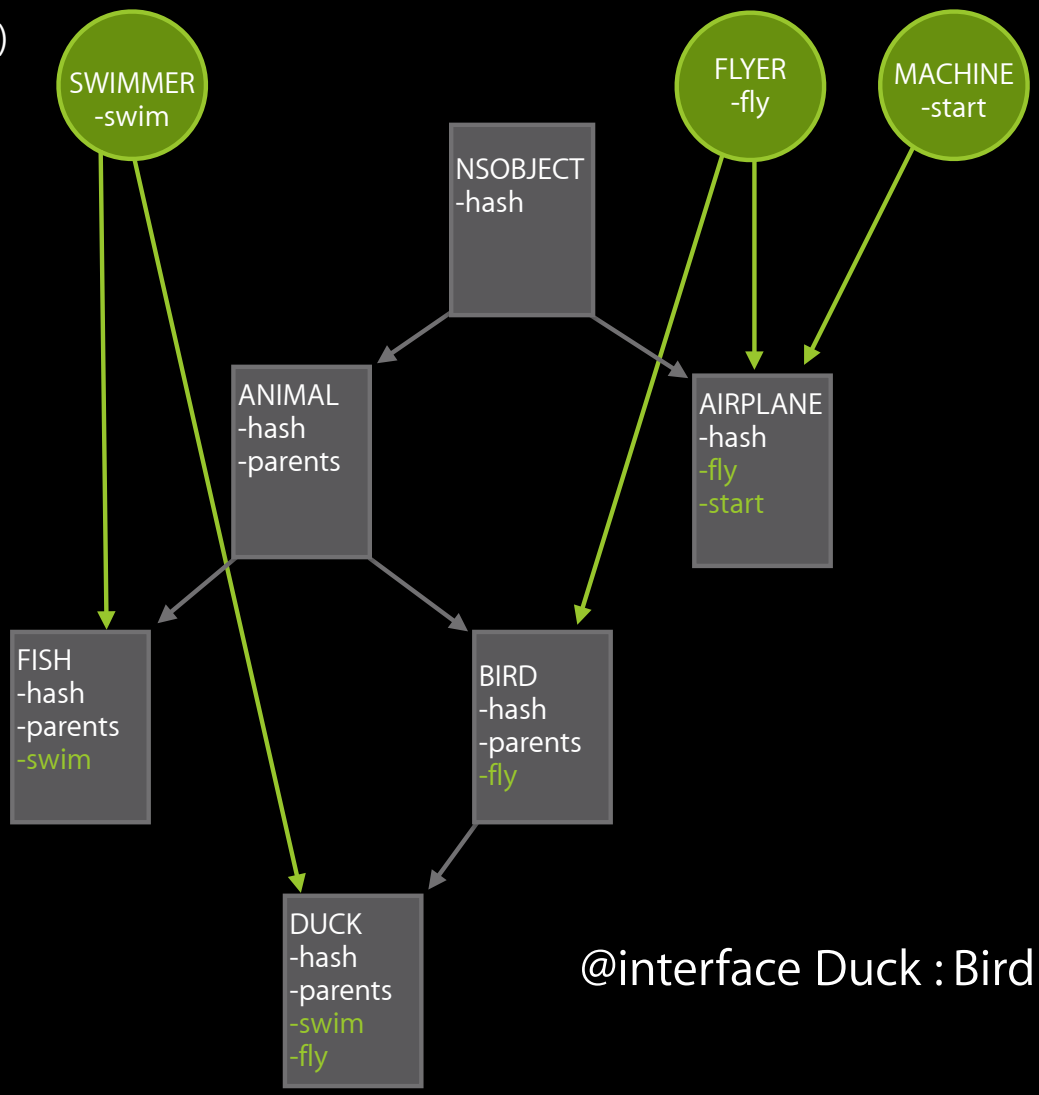
Java, C++, C#	Objective-C
member variable	instance variable, ivar
member function	method, instance method, -method, dash method
static method	class method, "plus method", +method, plus method
static variable	(global variable)
interface *doing*	@protocol *doing*
class *aclass*	@interface *aclass*
operator **new**	+alloc
~destructor	-dealloc

Protocols (Java: interface)

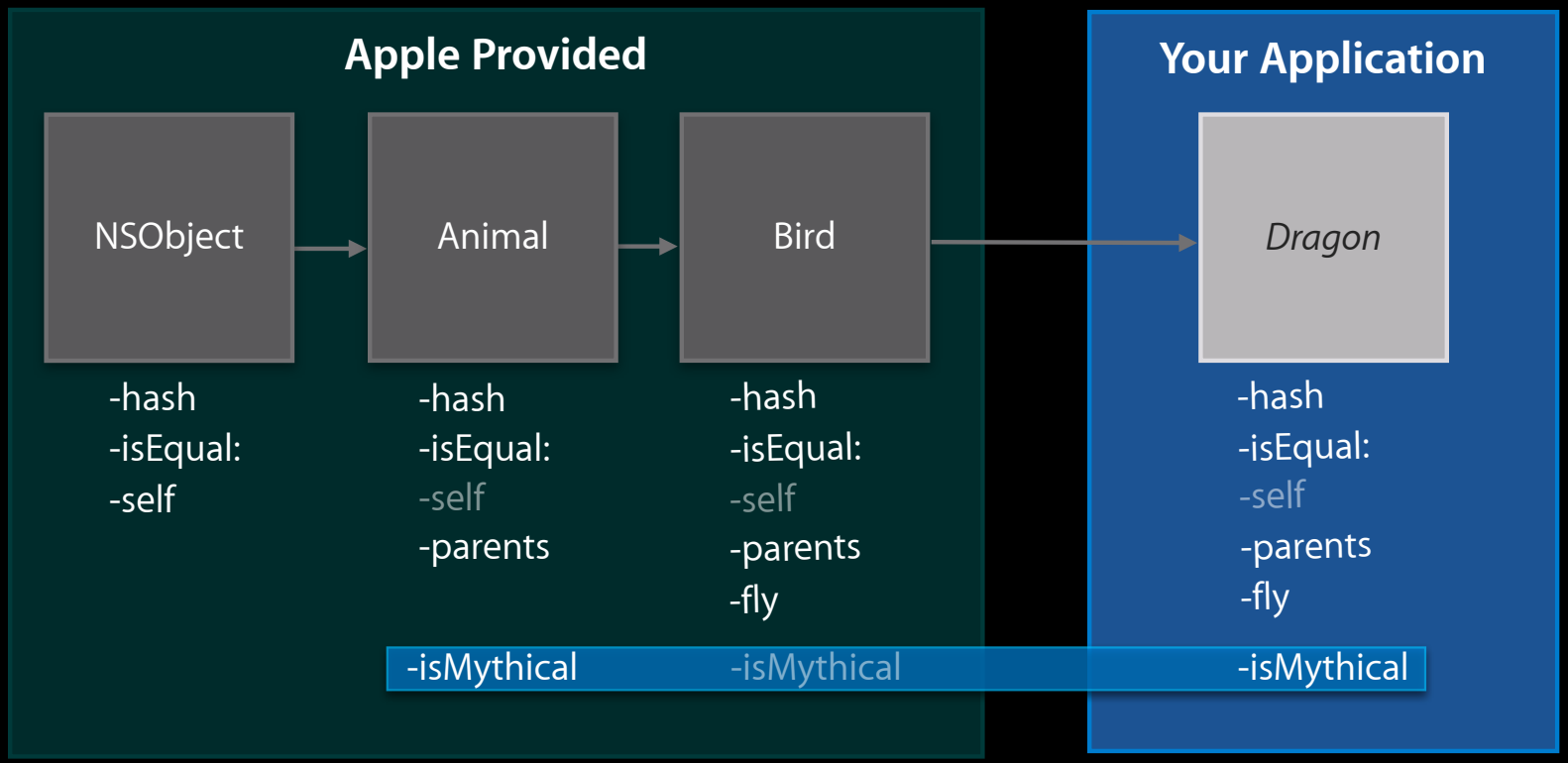
multiple-inheritance of abstract methods

Classes

single-inheritance of instance variables



@interface Duck : Bird <Swimmer, Flyer>



Objective-C Category:

```
@interface Animal (MythicalExtra)
-(BOOL)isMythical;
@end

@implementation Animal (MythicalExtra)
-(BOOL)isMythical { return NO; }
@end
```

```
@implementation Dragon (MythicalExtra)
-(BOOL)isMythical { return YES; }
@end
```

Categories

```
@interface Animal (MythicalExtra)
-(BOOL)isMythical;
@end
```

```
@implementation Animal (MythicalExtra)
-(BOOL)isMythical { return NO; }
@end
```

- Add behavior to any class
  - Use judiciously!
- Act like normal methods
  - Can call [super ...]
- With a little code can add data
  - Use Associative References
- Also can partition implementation
  - Access to @private allowed
  - Only within App/Framework!

Memory Management

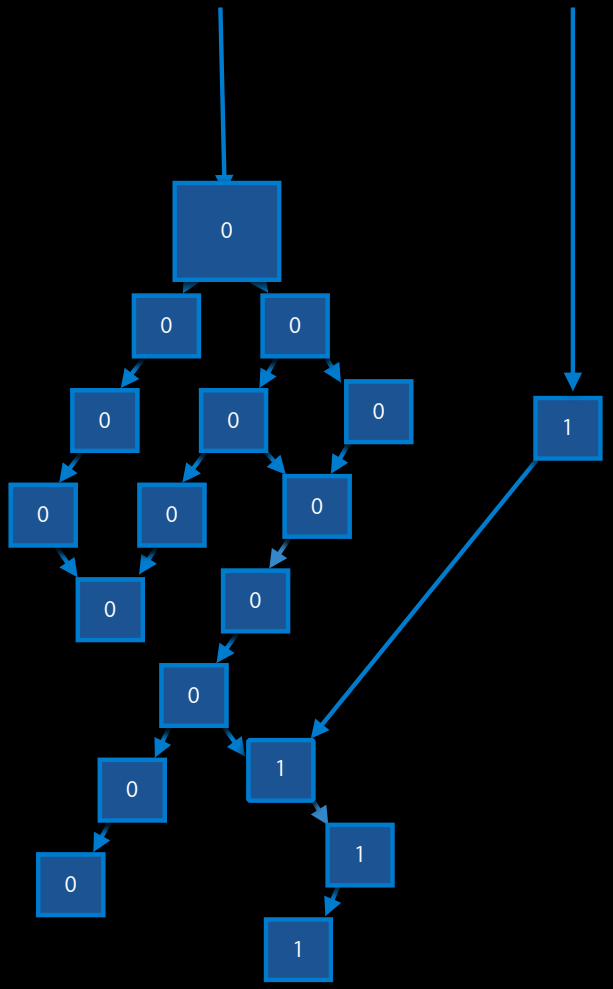
Memory Management: The Basics

- Motivation: No crashes! No leaks!

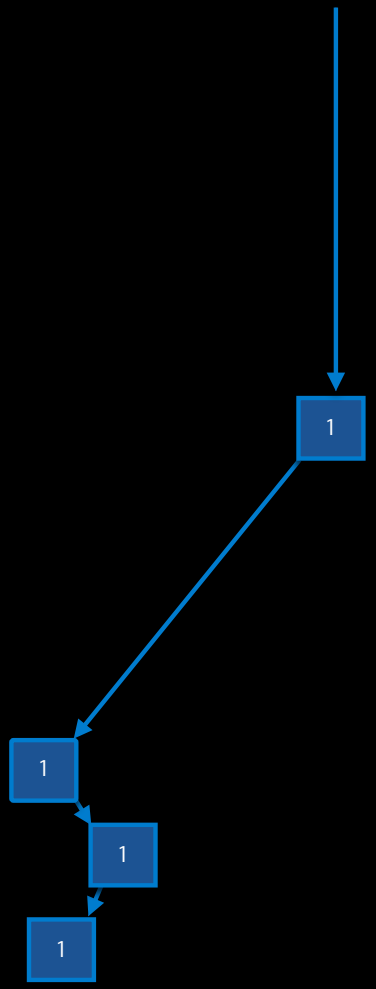
- Cocoa Touch system originally designed for eight-megabyte systems!
- Memory Management starts at the design phase
  - Object ownership is designed as Directed Acyclic Graph
  - Ownership arises from simple pattern
    - **Only** +alloc, -init*XXX*, -retain, -copy, +new*XXX* **create/transfer ownership**.

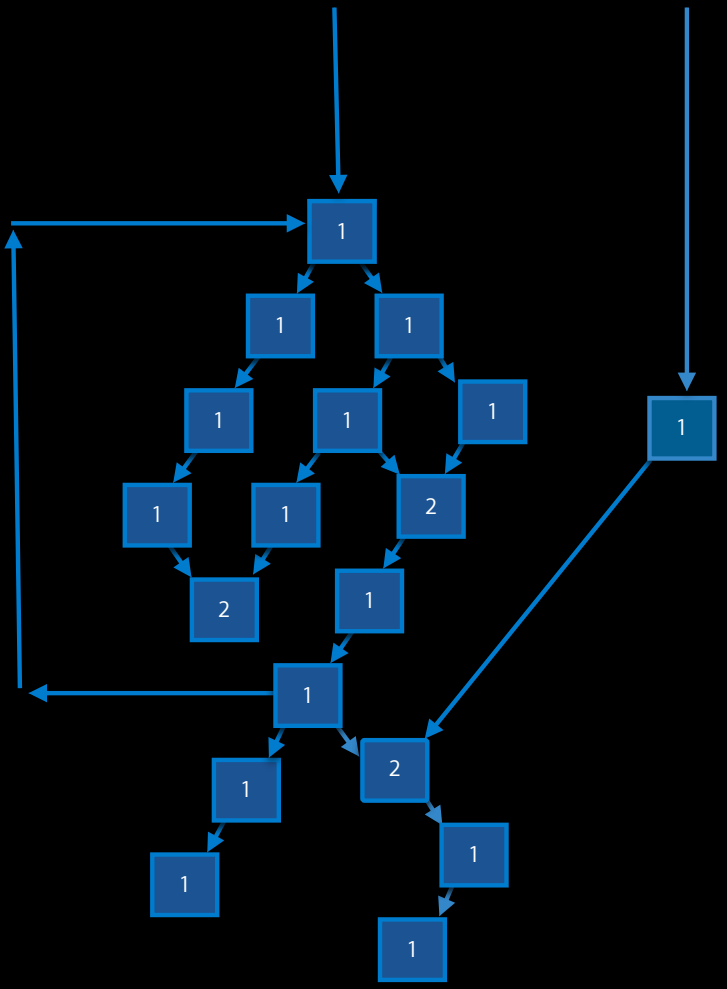
- Apple LLVM Static Analyzer helps you follow simple rules
- Apple Instrument Application measures memory behavior

Directed Acyclic Graph



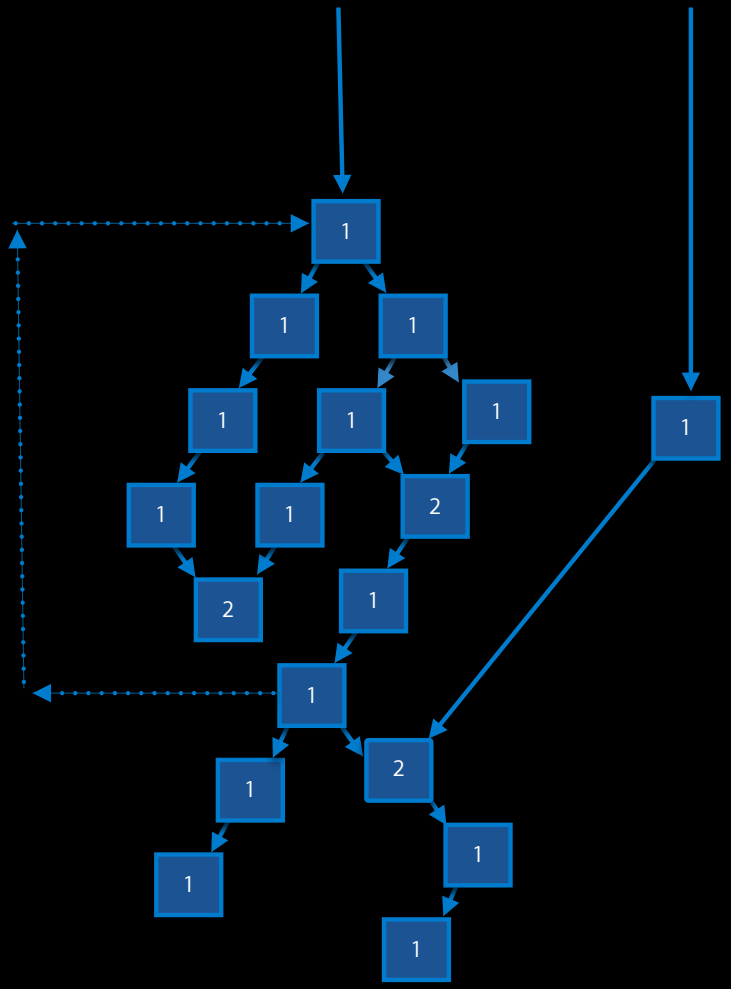
Directed Acyclic Graph





Cyclic Graph

- "Up" (back) pointer creates loop
- Objects never go away!



Simple Rules

- Instance variables are always either
  - Retained
  - Not retained
- Downlinks are retained
- Uplinks are not retained

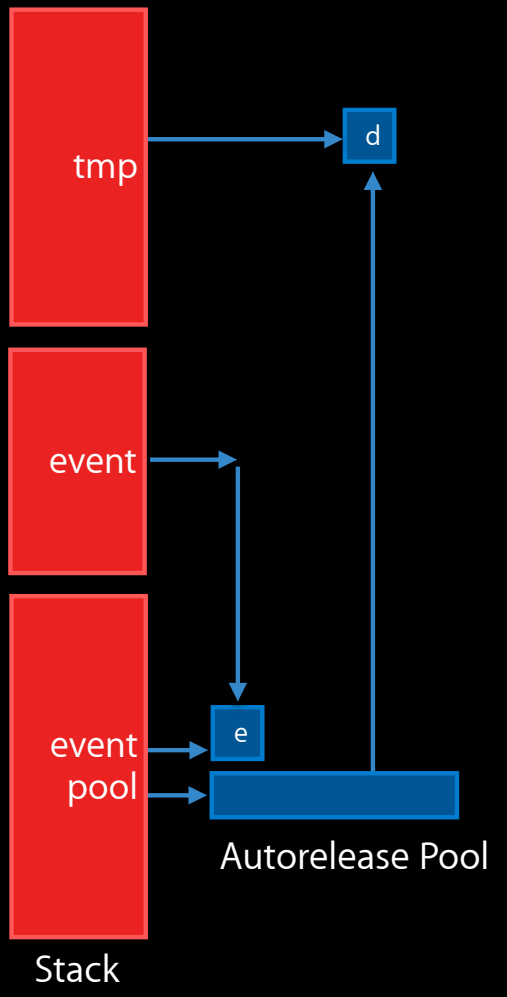
- Release old and
  - Retain new values,
    - Unless from -init*XXX* or -copy
- Autorelease new results

How Autorelease Pools Work

```
@implementation NSDate
...
+ (id) date {
   return [[[self alloc] init] autorelease];
}
...
@end
```

```
void process(NSEvent *event) {
    ...
    if ([event start] < [[NSDate date] ...])
}
```

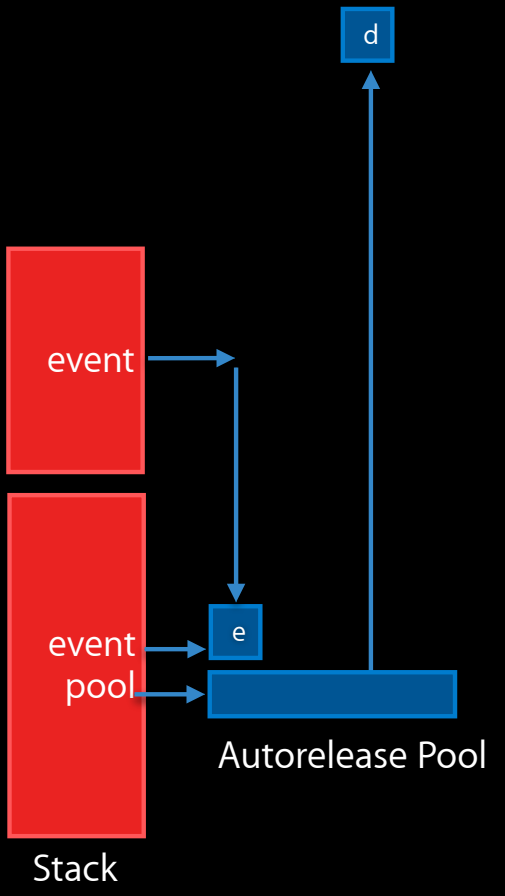
```
while (1) {
    NSEvent *event = getEvent();
    id pool = [NSAutoreleasePool new];
    process(event);
    [pool drain];
}
```



How Autorelease Pools Work

```
void process(NSEvent *event) {
    ...
    if ([event start] < [[NSDate date] ...])
}
```

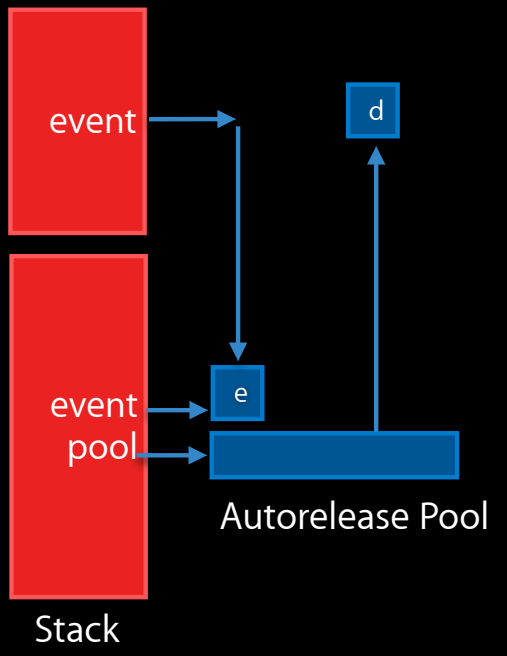
```
while (1) {
    NSEvent *event = getEvent();
    id pool = [NSAutoreleasePool new];
    process(event);
    [pool drain];
}
```



How Autorelease Pools Work

```
void process(NSEvent *event) {
    ...
    if ([event start] < [[NSDate date] ...])
}
```

```
while (1) {
    NSEvent *event = getEvent();
    id pool = [NSAutoreleasePool new];
    process(event);
    [pool drain];
}
```



Accessors

Cocoa Getter/Setter Pattern

CustomObject.h:

```
@interface CustomObject : NSObject {
    int balance;
}
- (int) balance;
- (void) setBalance:(int)newBalance;
@end
```

CustomObject.m:

```
@implementation CustomObject
- (int) balance { return balance; }
- (void) setBalance:(int)newBalance {
    balance = newBalance;
}
@end
```

Properties: Automatic Declaration and Methods

WWDC2010: 32-bit simulator uses modern runtime!

CustomObject.h:

```
@interface CustomObject : NSObject
@property int balance;
@end
```

CustomObject.m:

```
@implementation CustomObject
// @synthesize by default!!
@end  // Xcode 4 LLVM 2.0 Compiler!!
```

Advanced Objective-C and Garbage Collection Techniques

Pacific Heights
Friday 11:30AM

Property Attributes

Attributes define allowable behaviors

```
@interface CustomObject : NSObject
@property(readonly) int balance;
@end
```

```
@interface SuperCustom : CustomObject
@property(readwrite) int balance;
@end
```

All Attributes

getter=getBalance (and/or) **setter=markBalance:**	Custom method names
assign (or) **retain** (or) **copy**	Object ownership policy
nonatomic	Single-threaded only
readonly (or) **readwrite**	Getter only (or) both, can be changed by subclass

Property Implementations

You can explicitly code any or all parts of an @property

```
@interface CustomObject : NSObject {
    int secretBalance;
}
@property int balance;
@end
```

```
@implementation CustomObject
- (int) balance { return secretBalance; }
- (void) setBalance:(int)newBalance {
    secretBalance = newBalance;
}
@end
```

Property Implementations

Can designate backing instance variable with @synthesize

```
@interface CustomObject : NSObject {
    int secretBalance;
}
@property int balance;
@end
```

```
@implementation CustomObject
@synthesize balance=secretBalance;
@end
```

Deferred Implementation

Must use @dynamic

```
@interface CustomObject : NSObject
@property int balance;
@end
```

```
@implementation CustomObject
@dynamic balance;
@end
```

@property(copy) Getter/Setter Pattern

```
- (NSString *) title {
  @synchronized(self) {
    return [[title retain] autorelease];
  }
}
- (void) setTitle:(NSString *)newTitle {
  @synchronized(self) {
    NSString *tmp = [newTitle copy];
    [title release];
    title = tmp;
  }
}

- (void) dealloc {
    [title release];   //  or self.title = nil;
    [super dealloc];
}
```

Cocoa Patterns

Selectors

- Selectors are data structures that represent a method "slot" name
- The *id* object type allows any message to be sent without warning
- Can ask *respondsToSelector:*

```
- (void) aMethod:(id)object {
    if ([object respondsToSelector:@selector(fred)])
            [object fred];
}
```

Delegation—Your Part

```
@interface MyDelegate <UIActionSheetDelegate>
...
@end
```

```
@implementation MyDelegate
- (void)setUp {
   uiactionsheet.delegate = self;
}
- (void)willPresentActionSheet:(UIActionSheet *)as {
   ...
}
...
```

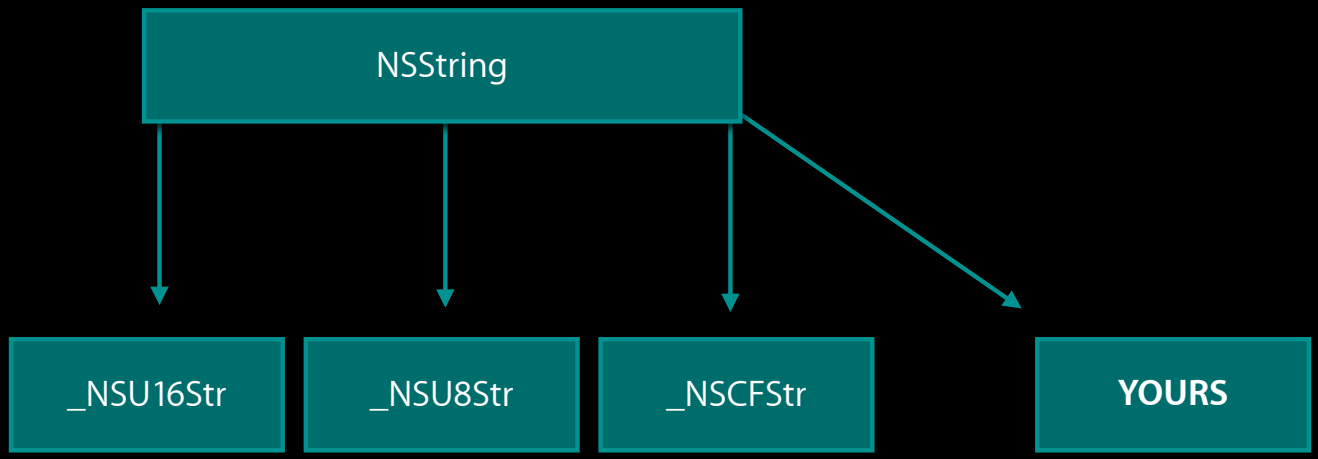
Delegation—UIKit Part

```
@protocol UIActionSheetDelegate <NSObject>
@optional
- (void)willPresentActionSheet:(UIActionSheet *)as;
- (void)didPresentActionSheet:(UIActionSheet *)as;
...
@end
```

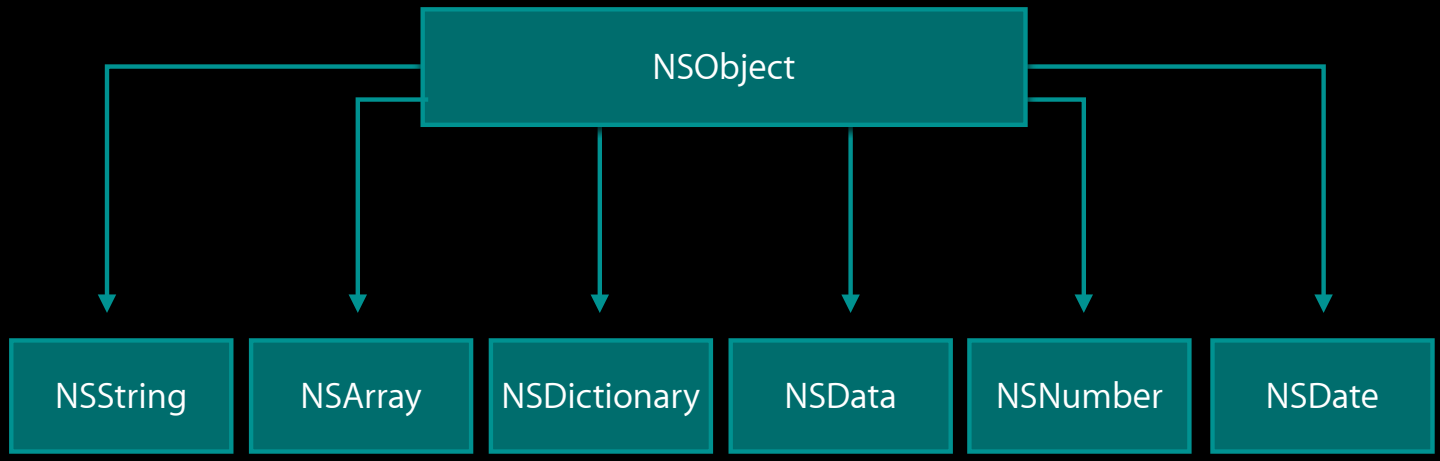
```
@interface UIActionSheet : UIView
@property(nonatomic,assign)
                  id<UIActionSheetDelegate> delegate;
...
@end
```

Class Clusters

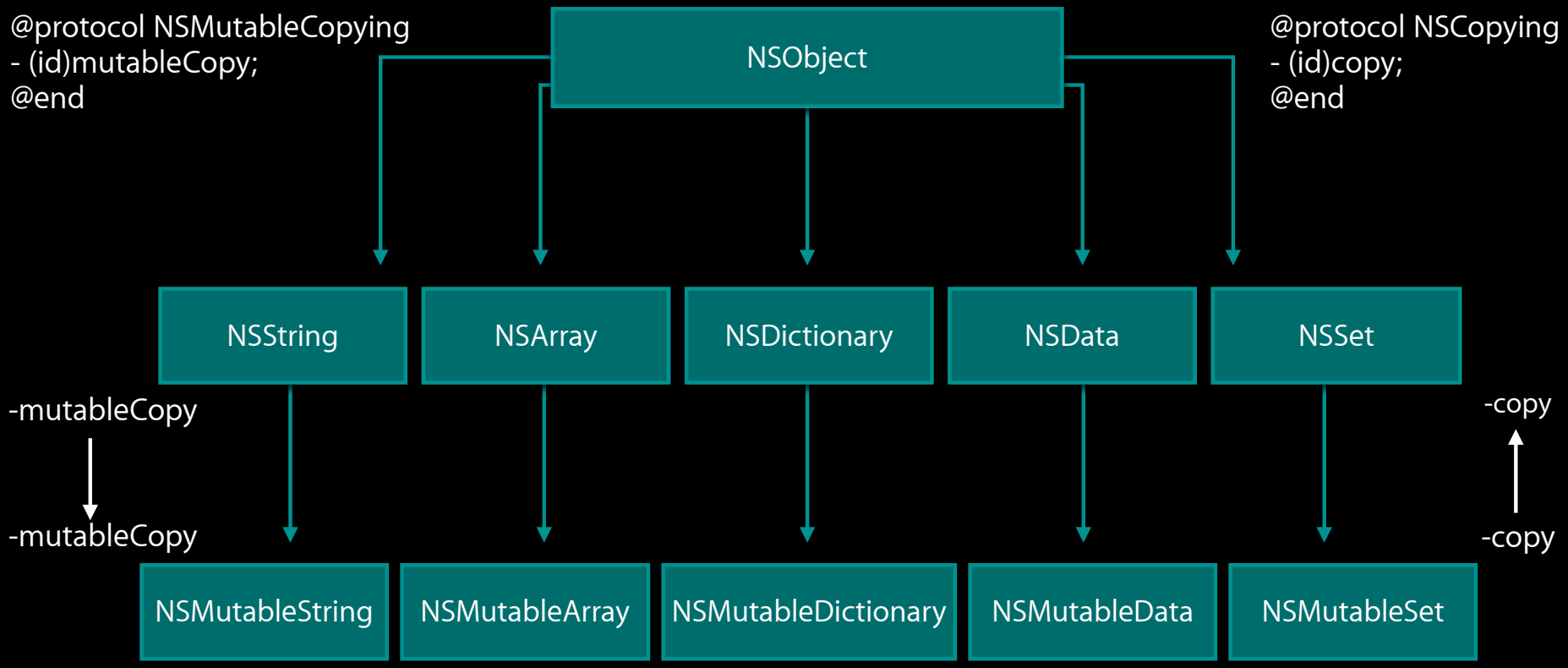
- Behavior specified in abstract super classes
- Private concrete implementations



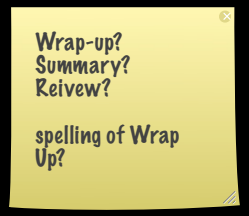
“Property List” Abstract Value Classes



Abstract Mutable Value Class Pattern



Wrap-up



- Map and Go With What You Know
  - Introduced Objective-C Terminology for **Common** Concepts
- Introduced Objective-C **Uncommon** Ideas
  - Blocks
  - @properties—let the compiler write your accessors!
  - Categories—add behavior (methods!) to any class
  - Selectors, Delegates, @optional
- Discussed Cocoa Touch **Patterns**
  - Memory Management—Retain, Release, Autorelease
  - Mutability, Class Clusters, "PLists"

Related Sessions and Labs

Introducing Blocks and Grand Central Dispatch on iPhone	Russian Hill Wednesday 11:30AM
API Design for Cocoa and Cocoa Touch	Marina Thursday 4:30PM
Advanced Objective-C and Garbage Collection Techniques	Pacific Heights Friday 11:30AM

Objective-C and Garbage Collection Lab	Developer Tools Lab A Thursday 2:00PM

More Information

Michaelopolis Jurewitz
Developer Tools Evangelist
jurewitz@apple.com

Apple Developer Forums
http://devforums.apple.com

Q & A

WWDC10