CS 696 Mobile Application Development
Fall Semester, 2010
Doc 4 Memory Management & Class Features
Sep 7, 2010

References

The Objective-C 2.0 Programming Language, http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/ObjectiveC/Introduction/introObjectiveC.html#//apple_ref/doc/uid/TP30001163

Memory Management

Memory Management

All objects allocated on heap

Objects must be deallocated - added back to free list

Garbage Collection

- Mac Desktop machines only

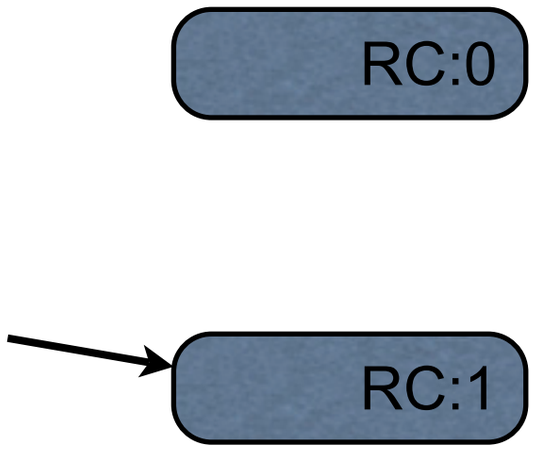
Reference Count

- Only option for iOS devices (iPhone/iPod Touch/iPad)

Reference Counting - Basic Idea

Object's retain count

number of references that what to retain object



When retain count is zero
an object is reclaimed

Note when an object is reclaimed (added back to the free list for the heap) its contents are not changed. So if you still access it then it still may appear to be valid.

Reference Counting - Questions

How does an object know

- when a reference points to it?
- when a reference stops pointing to it?

Reference Counting - Objective C Answer

How does an object know

- when a reference points to it?
- when a reference stops pointing to it?

You tell the object

Retain Count - Telling the Object

Method	Result
alloc	set retain count to 1
new	set retain count to 1
retain	increase retain count by 1
release	decrease retain count by1

Example

```
Rectangle * sample = [[Rectangle alloc] init]; // retain count 1

[sample setWidth:4];
[sample setHeight:5];
area = [sample area];
NSLog(@"%@",sample);

[sample release]                    // retain count 0

//sample is reclaimed
```

Retain Count 0 & Messages

```
Rectangle * sample = [[Rectangle alloc] init]; // retain count 1
[sample release]                    // retain count 0

[sample setWidth:4];      // What happens here?
                                    // If lucky program crashes
```


Note when an object is reclaimed (added back to the free list for the heap) its contents are not changed. So if you still access it then it still may appear to be valid. This can lead to some hard to find bugs as some point the memory may be reused and the data will be changed. But this will happen far from the point in the code where the actual problem is located.

Owning an Object

Way to thing about handling retain count

If you own the object must release object

If you don't own the object do not release the object

Owning An Object

You own an object you it create using:

- method starting with new
- method starting with alloc
- method containing copy

You own an object if you sent it the message

- retain

When done with an object you own release it using

- release
- autorelease

If you don't own the Object

Don't release it

Example

```
NSString * mine = [[NSString alloc] initWithString:@"cat"];
NSString * notMine = [NSString stringWithFormat:@"size = %i", 10];
NSString * alsoMine = [notMine copy];
[mine release];
[alsoMine release];
```


Who owns notMine? The above code segment does not so it does not release it

Issues with retain count

Method return values

Instance Variables

How to return new object

```
-(NSString*) description {
	NSString* description;
	description = [[NSString alloc] initWithFormat:@"Person: %@", name];

	return description;
}
```

The description method owns description string

But did not release it

Since the calling code does not own the returned string it will not release it. So this is a memory leak.

How to return new object

```
-(NSString*) description {
        NSString* description;
        description = [[NSString alloc] initWithFormat:@"Person: %@", name];
        [description release];
        return description;
}
```

The description method owns description string & released it

This does not work - string no longer exists - call can't use it

Actually the return value maybe valid for a while, so you might get away with this for a bit but it will eventually cause you serious problems

How to return new object

```
-(NSString*) description {
        NSString* description;
        description = [[NSString alloc] initWithFormat:@"Person: %@", name];
        [description autorelease];
        return description;
}
```

This works

String released, but later

Caller gets valid object, can retain if needed

How does autorelease work?

[description autorelease]

autorelease adds object to NSAutoreleasePool

NSAutoreleasePool owns the object and releases it when drained

```
int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

    // Add code here
       [pool drain];
    return 0;
}
```

NSAutoreleasePool & iPhone Apps

iPhone apps are event driven

Each user event (touch, keystroke, etc) has its own NSAutoreleasePool

autoreleased objects are released after your code handles the event

So you always have an NSAutoreleasePool

String Example Revisited

```
NSString * mine = [[NSString alloc] initWithString:@"cat"];
NSString * notMine = [NSString stringWithFormat:@"size = %i", 10];
NSString * alsoMine = [notMine copy];
[mine release];
[alsoMine release];
```

notMine has retain count 1

NSAutoreleasePool will release it

Multiple Owners

An object can have multiple owners

Each owner needs the object to stay around

When last owner releases object it is reclaimed

When to own an object

When you

- have a reference to an object and
- need to keep the object

For example

- Instance variables
- Globals (static)

Owning Objects - Instance variable

```
@interface Person : NSObject {
      NSString* name;
      int age;
}

-(void) setName: (NSString*) newName;
-(NSString*) name;
-(void) setAge: (int) newAge;
-(int) age;
@end
```

Owning & Setter Methods

```
@implementation Person

- (void) setAge:(int)newAge {
      age = newAge;                  // OK
}

-(void) setName: (NSString*) newName {
      name = newName;                // Don't do this

   // 3 major problems
   // Memory leak
   // Some one else owns name
   // Other code can access name
}
```

Owning & Setter Methods

```
-(void) setName: (NSString*) newName {
        if (name != newName) {
                [name release];
                name = [newName retain];
        }
}
```

Memory leak fixed

Object owns new name

Owning & Setter Methods

```
-(void) setName: (NSString*) newName {
        if (name != newName) {
                [name release];
                name = [newName copy];
        }
}
```

Memory leak fixed

Object owns new name

Object has own copy

Releasing Instance Variables

```
@implementation Person

    -(void) dealloc {
                [name release];
                [super dealloc];
    }
```


When an object is released it is sent the dealloc method

[[Rectangle alloc] initWithWidth: 3 height: 4]

Retain & Constructors

```
@interface Rectangle : NSObject { int width; int height; }

- (id) initWithWidth: (int) newWidth height: (int) newHeight;
@end
```

```
@implementation Rectangle

- (id) init { return [self initWithWidth: 0 height: 0]; }

- (id) initWithWidth: (int) newWidth height: (int) newHeight {
        if (self = [super init]) {
                width = newWidth;
                height = newHeight;
        }
        return self;
}
```

[[Rectangle alloc] initWithWidth: 3 height: 4]

Retain & Constructors

```
@interface Rectangle : NSObject { int width; int height; }
+ (id) square: (int) length;
- (id) initWithWidth: (int) newWidth height: (int) newHeight;
@end
```

```
@implementation Rectangle

+ (id) square: (int) length {
    id square = [[self alloc] initWithHeight: length width: length];
    [square autorelease];
    return square;
}
```

malloc

If you allocate memory using malloc, calloc, etc

Release it using free

Review

When own object insure you release it

Own object when you want to keep it

Don't release objects you don't own

Class Features

Properties

```
#import <Foundation/Foundation.h>

@interface Person : NSObject {
    int age;
}

@property int age;
@property NSString* name;
@end
```

```
#import "Person.h"


@implementation Person

@synthesize name;
@synthesize age;

-(void) dealloc {
    [name release];
    [super dealloc];
}

@end
```


@property declares the propery, @synthesize creates accessor methods. In older version had to also declare the instance variable. Do not have to declare the instance variable for property, compiler will do it for you. In future version will not have to use @synthesize

Using Properties

```
Person* you = [[Person alloc] init];
you.age = 10;
[you setAge:12];
int yourAge = you.age;
yourAge = [you age];
```

Property Attributes

```
@interface Person : NSObject {
      NSString* fullName;
      int age;
}

@property int age;
@property (copy, readonly, getter = name) NSString*  fullName;

- (id) initWithName: (NSString*) name;

@end
```

Implementation

```
@implementation Person

@synthesize fullName;
@synthesize age;

-(void) dealloc {
        [fullName release];
        [super dealloc];
}

- (id) initWithName: (NSString*) name {
        if (self = [super init]) {
                fullName = [name copy];
        }
        return self;
}
@end
```

Using the Property

```
Person* you = [[Person alloc] initWithName:@"Sam"];
you.age = 10;
[you setAge:12];
int yourAge = you.age;
yourAge = [you age];
NSString* name = you.name;
name = [you name];
```

Properties

Accessor Method names	Writability	Setter Semantics
getter=getternameㅤ	readwrite (default)	assign (default)
setter=setternameㅤ	readonly	retain
		copy

Accessor methods & Memory management

```
@interface Person : NSObject {
      NSString* name;
}
@property  NSString*  name;

- (id) foo;

@end
```

```
#import "Person.h"

@implementation Person

@synthesize name;

- (id) foo {
      NSString * bar;
      some code that create bar string

      name =  bar;
}
```


Memory leak

Accessor methods & Memory management

Use accessor methods internally
to modify instance variable to avoid
memory leaks

```
#import "Person.h"

@implementation Person

@synthesize name;

- (id) foo {
      NSString * bar;
      some code that create bar string

      [self name: bar];
}
```

Categories

Add methods to classes

NSString-Extra.h

```
@interface NSString (NSString_Extras)

- (NSString*) reverse;

@end
```

NSString-Extra.m

```
#import "NSString-Extras.h"

@implementation NSString (NSString_Extras)

- (NSString*) reverse {
    int size = [self length];
    unichar reversedChars[size];
    for (int k = 0;k < size;k++) {
        reversedChars[k] = [self characterAtIndex: size - k - 1];
    }
    NSString * reversed = [[NSString alloc] initWithCharacters:reversedChars length:size];
    [reversed autorelease];
    return reversed;
}

@end
```

Using the new NSString method

```
#import "NSString-Extras.h"

int main (int argc, const char * argv[]) {
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];

   NSString * result = [@"rat" reverse]);
      [pool drain];
    return 0;
}
```

Ways to use Categories

Add methods to existing classes

Allow you to define a class in multiple files

- Each file containing related methods

Extensions - Empty Categories

```
@interface ClassExtensionExample :
      NSObject {
}

- (void)normalMethod;
@end

@interface ClassExtensionExample ()
-(void) headerExtension;
@end
```

```
#import "ClassExtensionExample.h"
@interface ClassExtensionExample ()
-(void) implementationExtension;
@end

@implementation ClassExtensionExample
- (void)normalMethod {
    NSLog(@"normal");
}

- (void)headerExtension {
    NSLog(@"headerExtension");
}

- (void)implementationExtension {
    NSLog(@"implementationExtension");
}
@end
```


Sometimes used to implement methods intended to be privately used in class

Protocols

Like Java's interface

Phones.h

```
@protocol Phones
- (NSString*) mobile;
- (NSString*) work;
@end
```

```
#import "Phones.h"

@interface Person : NSObject <Phones>{
        NSString* fullName;
        int age;
}

@property int age;

- (id) initWithName: (NSString*) name;

@end
```

Using Protocol

```
#import "Person.h"

@implementation Person

@synthesize age;

- (NSString *) mobile {
    return @"619-999-9999";
}

- (NSString *) work {
    return @"619-999-9999";
}

etc.
```

```
Person* you = [[Person alloc] initWithName:@"Sam"
NSString* cellNumber = [you mobile];
id <Phones> test = you;
```

Protocol Details

Can define methods & properties

optional & required (default) methods

```
@protocol MyProtocol
- (void)requiredMethod;

@optional
- (void)anOptionalMethod;
- (void)anotherOptionalMethod;

@required
- (void)anotherRequiredMethod;
@end
```

Blocks

```
int (^increase)(int) = ^(int amount) { return amount + 1;};
int result = increase(10);
```

```
typedef int (^adder)(int);
int increment = 3;
adder increase = ^(int amount) { return amount + increment;};
increment = 20;
int result = increase(10);
// result == 13
```