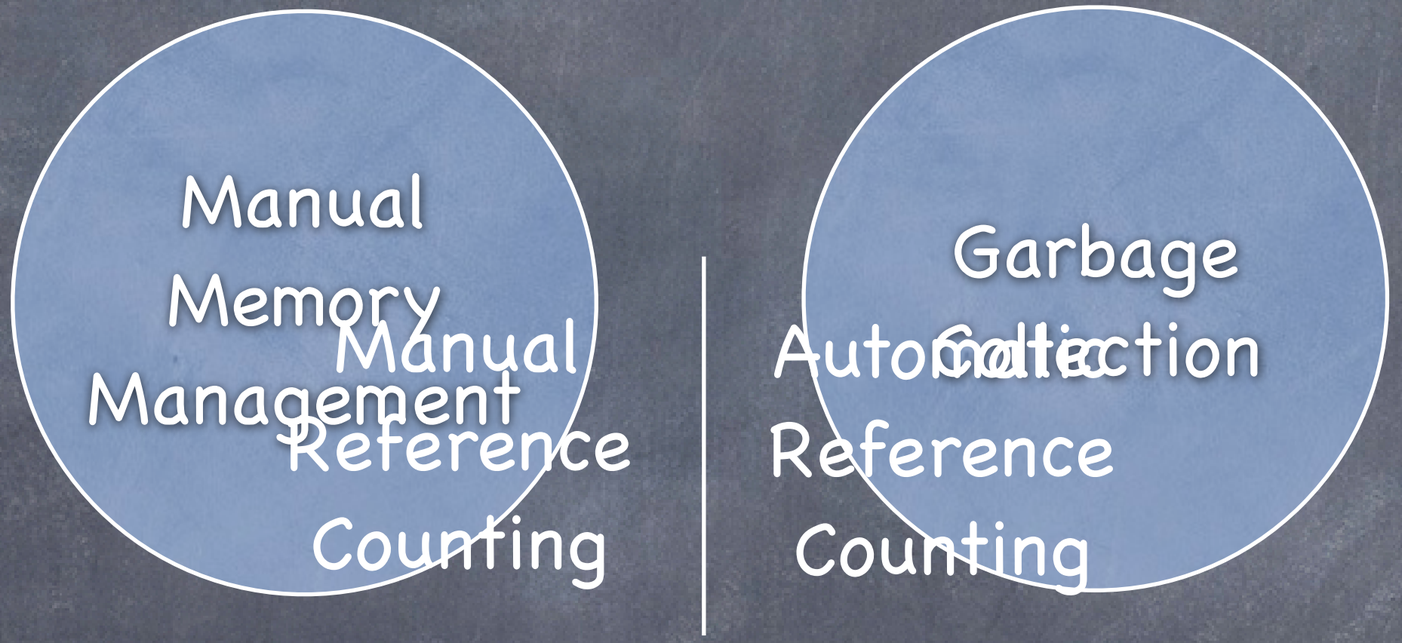
Objective-C Memory Management

Matthias Tretter, @myell0w

Overview



e.g. C, C++, Objective-C

e.g. Java, C#, JavaScript, Haskell, Objective-C (deprecated)

Garbage Collection (GC)

- Known from Java
- No need to manually manage memory
- Garbage Collector does it for you - at run time!

„Reference Counting is a technique of storing the number of references, pointers, or handles to a resource such as an object, block of memory, disk space or other resource“

http://en.wikipedia.org/wiki/Reference_counting

Manual Reference Counting

Source Code

Reference Count

```
NSObject *o = [[NSObject alloc] init];
[o retain];
[o retain];
[o release];
[o release];
[o release];
```

3

2

1

[o dealloc];

Object o

Automatic Reference Counting (ARC)

- Somewhere in-between reference counting and GC
- No need to write memory management code
- BUT knowledge of memory management rules needed
- compile-time feature
- handles retain/release calls for you

Ownership

- Object lifecycle is determined by terms of ownership
- If you want to keep an object around: take ownership
- If you don`t need it anymore: relinquish ownership
- Object gets destroyed when there`s no owner anymore

Ownership Axioms

1. An object will exist in memory so long (but no longer) as at least one object maintains ownership of it.
2. To keep an object, you must take ownership of it. If you are done with an object, you must relinquish ownership of it.
3. More than one object may share ownership of a given child object.

Acquire Ownership

1. Allocate an object

```
object = [[Type alloc] init]
object = [[Type alloc] initWithSomeParameters:...]
object = [Type new]
```

Acquire Ownership

2. Copy an existing object

```
object = [anotherObject copy]
object = [anotherObject mutableCopy]
```

Acquire Ownership

3. Explicitely request ownership

```
// assign to a strong variable/property
// @property (nonatomic, strong) NSObject *object;
self.object = anotherObject
```

Acquire Ownership Summary

1. Allocate an object

```
object = [[Type alloc] init]
```

2. Copy an existing object

```
object = [anotherObject copy]
```

3. Explicitely request ownership

```
// @property (nonatomic, strong) NSObject *object;
self.object = anotherObject
```

Relinquish Ownership

1. Remove the strong reference to the owned object

```
self.strongObject = nil
```

2. There is no step 2

„If you don't claim ownership of an object, you can't expect it to be around for any longer than its current scope."

Scope

```
- (void)methodScopeExample {
    // scope = method

    if (someBOOL) {
        // scope = if-block
    }

    // scope = method

    {
        // scope = anonymous block
    }

    // scope = method
}
```

„If you don't claim ownership of an object, you can't expect it to be around for any longer than its current scope."

Examples

```
- (void)someMethod {
    // creates ownership, but only for the method's scope
    id localObject = [SomeClass new];

    // ... do your stuff with localObject

    // ARC automatically relinquishes ownership of
    // localObject for us, because self didn't
    // take Ownership
    return;
}
```

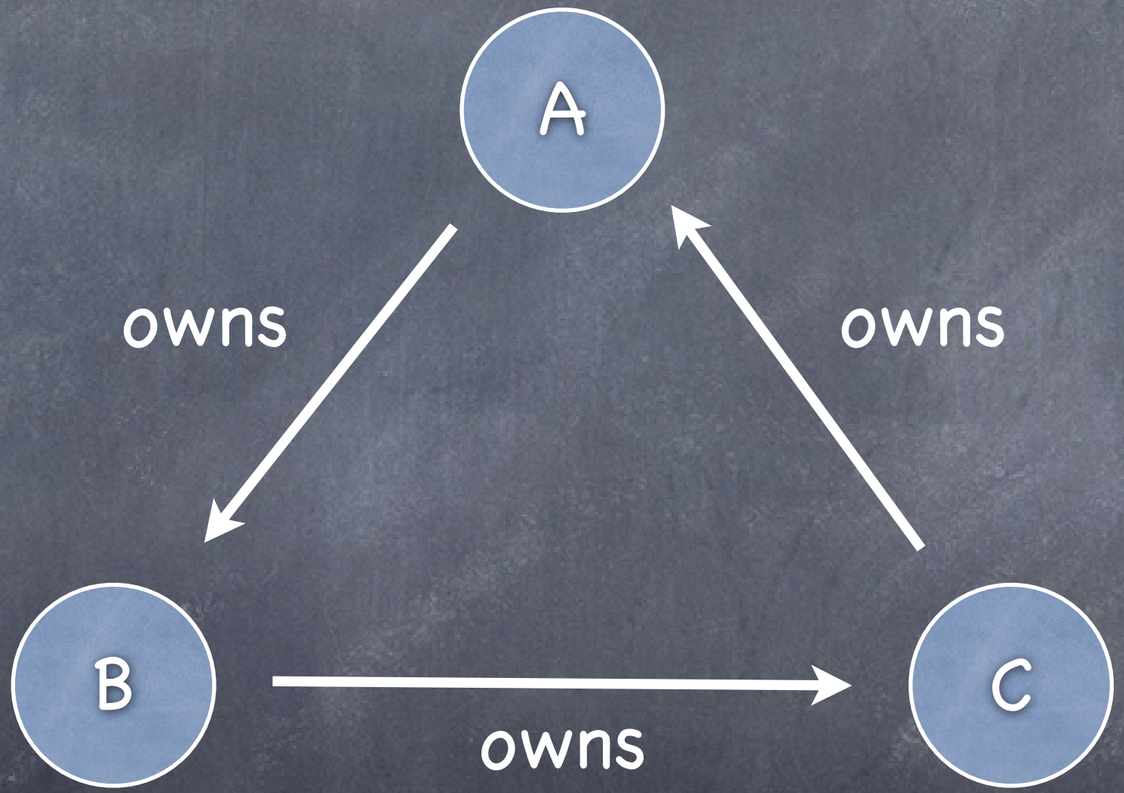
Examples

```
- (void)setupState {
    // In the below case, we assign the object to a strong
    // property, thus taking ownership.
    self.strongProperty = [SomeClass new];

    // ... etc.

    // Even though the -new method also comes with
    // ownership, it's local like in the last example,
    // so we get the intended behavior of a single
    // ownership. ARC figures it out for us.
}
```

Examples

```
- (void)setChild:(Child *)child {
    self.child = child;

    // @property (nonatomic, strong) id parent;
    self.child.parent = self;
}
```

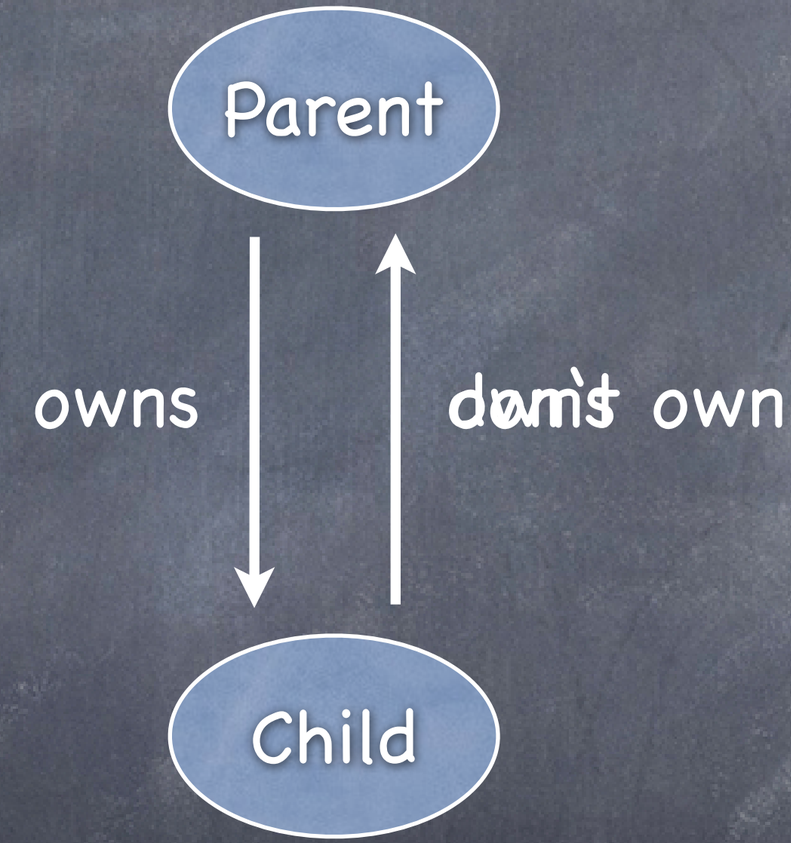
Ownership Cycles



Storage Classes

```
@property (nonatomic, strong) id strongObject

@property (nonatomic, copy) id strongObject

@property (nonatomic, weak) id weakObject

// @property (nonatomic, unsafe_unretained) id weakObject
```

Ownership Cycles



strong vs weak

- strong "keep this in the heap until I don't point to it anymore"
  I won't point to it anymore if I set my pointer to it to nil.
  Or if I myself am removed from the heap because no one strongly points to me!
- weak "keep this as long as someone else points to it strongly"
  If it gets thrown out of the heap, set my pointer to it to nil automatically (if user on iOS 5 only).
- This is not garbage collection!
  It's way better. It's reference counting done automatically for you.
- Finding out that you are about to leave the heap
  A special method, dealloc, is called on you when your instance's memory is freed from the heap.
  You will rarely ever have to implement this method. It's "too late" to do much useful here.

```
- (void)dealloc
{
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}
```

Further Sources

- Transitioning to ARC Release Notes, Apple
  http://developer.apple.com/library/mac/#releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html
- Modern Cocoa Memory Management, Jason Brennan
  http://nearthespeedoflight.com/article/modern_cocoa_memory_management
- Understanding Automatic Reference Counting in Objective-C
  http://longweekendmobile.com/2011/09/07/objc-automatic-reference-counting-in-xcode-explained/