Objective-C For Professional Developers

Section 6
Memory Management

AboutObjects

www.aboutobjects.com

Management Schemes

- Garbage Collection
  - Leopard, Snow Leopard, Lion
- Manual Reference Counting
- ARC (Automatic Reference Counting) – iOS 5
  - Partialy back-compatible with 4.3 and 4.2 (they don't support zeroing weak references).

Reference Counting

- Implemented in Foundation framework
  - Methods declared in NSObject protocol
- The address of an object is a ***reference***
- Keeps track of the number of variables that currently store a reference to a given object

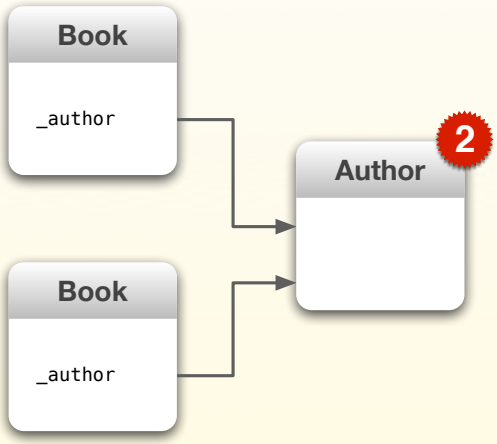
Conceptual Overview

- Each object maintains its own reference count
  - Initially **1**
- Tell the object to increment its count whenever you store another reference to it in an ivar or static variable
- Tell the object to decrement its count whenever you delete a reference to it in an ivar or static variable
- When count drop to **0**, object deallocates itself

Object Ownership

Book Objects With Shared Author Object

- Each Book object has a reference to the same Author object
- Author object's retain count reflects current number of references
- Author will deallocate itself when its retain count drops to zero



Methods Overview

- retain	Increments the receiver's reference count.
- retainCount	Returns the receiver's current reference count.
- release	Decrement's the receiver's reference count.
- autorelease	Adds the receiver to the current autorelease pool.
- dealloc	Deallocates the memory occupied by the receiver.

- retain

- Send an object a **retain** message if you need it to stay valid beyond the end of the current method or function.
- Increasing an object's retain count ensures it won't deallocate itself before you're done using it.
- Calls to **retain** most commonly seen in setter methods.

Sending -retain in Setter

- Argument has unknown retain count
- Sending retain message guarantees retain count won't drop to zero later

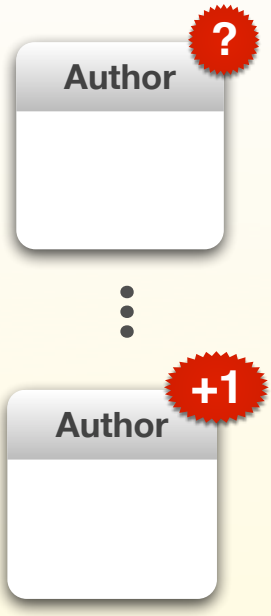
```
@implementation Book

//...

// NOTE: This implementation leaks.
//
- (void)setAuthor:(Author *)anAuthor
{
    _author = [anAuthor retain];
}

//...

@end
```



- release

- Send an object a **release** message to when you want it to decrement its retain count.
- Goal: balance calls to creation methods (+**alloc**, -**copy**, etc.) and calls to -**retain** with calls to -**release**.
  - Conceptually similar to balancing parentheses.
- Where possible, send -**release** message before closing curly brace of method that called +**alloc** or -**copy**.

Sending -release Message

- Call to **+alloc** returns an object with retain count of **1**
- **-setAuthor** method stores reference, manages retain count as necessary (in this case, sending **-retain** message).
- **-release** message balances **+alloc** to ensure count can eventually drop to zero; otherwise, object is leaked.

```
@implementation Book

//...
- (void)addAuthorWithFirstName:(NSString *)aFirstName
                      lastName:(NSString *)aLastName
{
    Author *newAuthor = [[Author alloc]
                         initWithFirstName:aFirstName
                         lastName:aLastName];
    [self setAuthor:newAuthor];
    [newAuthor release];
}
```

- autorelease

- Send **-autorelease** when you need an object to remain valid temporarily, but still need to balance a call to **+alloc**, **-copy**, **-retain** , etc. with a call to **-release**.
- For example, suppose an object has an instance variable of type **NSMutableArray *.**
  - You might want getter method to return a copy.
  - Avoids risk that two or more objects might end up accidentally modifying the same array.

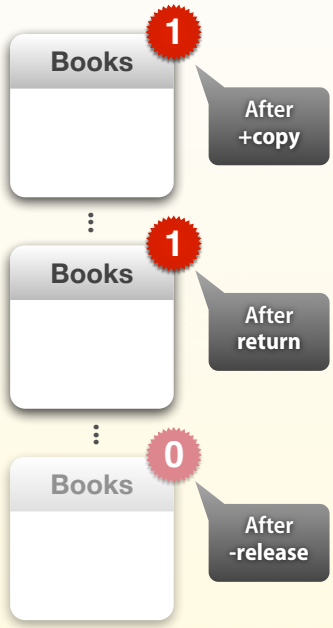
Sending -autorelease in Getter

- Call to **+copy** returns an object with retain count of **1**
- If the getter method sends it a **-release** message the object returned will already have deallocated itself.
- **-autorelease** tells the object to add itself to a pool of objects that will get a deferred **release** message later.

```
@implementation ListController

//...
- (NSArray *)books
{
    // Create immutable copy. Retain count will be 1.
    NSArray *copyOfBooks = [books copy];
    // Send autorelease to balance the call to +copy.
    [copyOfBooks autorelease];
    // Copy will remain valid to end of calling code.
    return copyOfBooks;
}
```

-release message will be sent after calling method completes



NSAutoreleasePool

- Special type of collection that contains objects that have received **autorelease** messages.
  - Maintains count per object
- When pool is deallocated, it sends **release** messages to all if the objects it contains.
  - Objects that have received more than one **autorelease** message will receive a corresponding number of **release** messages.

Nested Autorelease Pools

- You can create and release instances of NSAutoreleasePool wherever you like.
  - Can help keep memory footprint small.
  - Can also avoid noticeable delays caused by processing large pools.
- Typically useful at beginning and end of large loops.

Autorelease Pools in ARC

- When you compile with ARC enabled, it's illegal to instantiate an NSAutoreleasePool yourself.
- Instead use the new **@autoreleasepool** compiler directive.

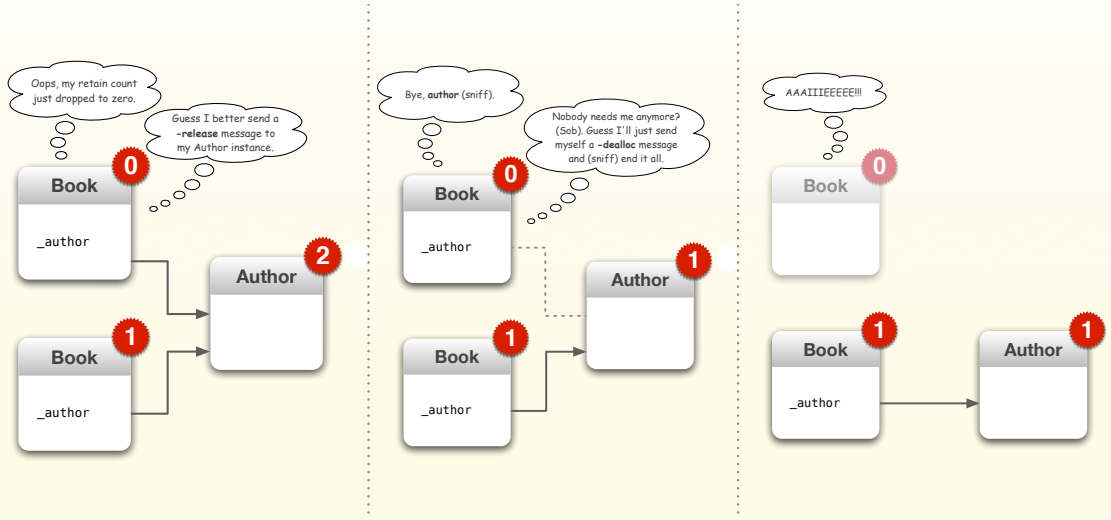
```
@autoreleasepool {
    ...
    NSLog(@"Do some autoreleased stuff...");
    ...
}
```

- dealloc

- You never send a **-dealloc** message.
- **-dealloc** called automatically by **-release** when retain count drops to zero.
- Override **-dealloc** if instances of your class will need to send **-release** messages to objects retained in ivars.
  - Provides a way to balance **-retain** and **-copy** messages sent in setter methods.

Deallocating a Book Object

Caution: viewer discretion advised.



Accessor Methods

Potential Issues in Setters

- Just assigning the new value isn't enough; setter method must send a **-release** message to current object in ivar before assigning new object.

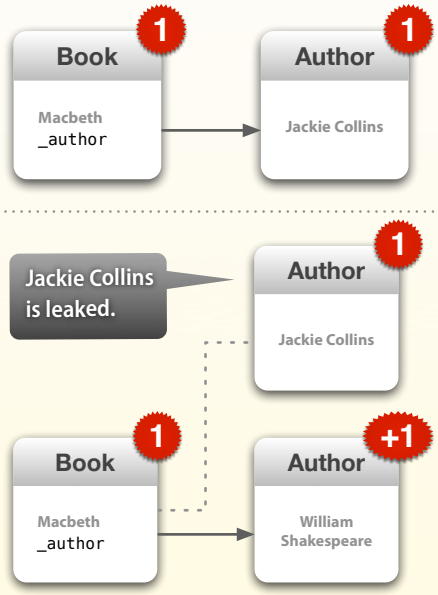
```
@implementation Book

//...

// NOTE: This implementation leaks.
//
- (void)setAuthor:(Author *)anAuthor
{
    _author = [anAuthor retain];
}

//...

@end
```



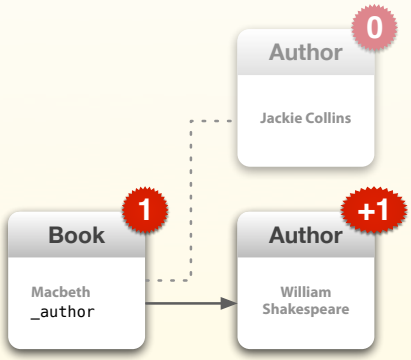
Implementing Setter Methods

- In addition to sending **-release** to current object, setter must avoid potential crasher bug if same object passed in twice in a row.

```
@implementation Book
//...

- (void)setAuthor:(Author *)anAuthor
{
    // If passed same object, do nothing.
    if (anAuthor == _author)
        return;

    // Send release to current instance.
    [_author release];
    // Replace current instance and
    // retain the new one.
    _author = [anAuthor retain];
}
```



For More Info…

- Apple's Memory Management Programming Guide
  - Detailed info, examples, and guidelines on memory management; a must-read for Cocoa and Cocoa touch developers.
- The NSObject Protocol Reference
  - Method-level documentation on core methods.