About Memory Management

Class A

with it. A well-written program uses as little memory as possible. In Objective-C, it can also be seen as a way of distributing ownership of limited memory resources among many pieces of data and code. When you have finished working through this guide, you will have the knowledge you need to manage your application's memory by explicitly managing the life cycle of objects and freeing them when they are no longer needed. Application memory management라는 것은 여러분의 프로그램이 동작할 때(runtime) 메모리를 할당하고, 그것을 사용하고, 사용을 끝마친 후 그것을 해제해주는 일련의 과정

Application memory management is the process of allocating memory during your program's runtime, using it, and freeing it when you are done

을 의미합니다. 잘 짜여진 프로그램은 메모리를 최대한 적게 사용하는 것이라 할 수 있습니다. Objective-c에선 수 많은 데이터와 코드에 제한된 메모리자원의 오너쉽(소유권)을 분배 하는 하나의 방법으로 볼 수 있습니다. Although memory management is typically considered at the level of an individual object, your goal is actually to manage object graphs. You want

to make sure that you have no more objects in memory than you actually need.

Class A

Class B

Class B

alloc/init release retain release Destroyed 2 2 Retain count =1 Destroyed copy release Class C Class C

(reference counting)이라 불리는 런타임환경과 연계하여 구현되는 모델을 사용하여 구현합니다. 2. In Automatic Reference Counting, or ARC, the system uses the same reference counting system as MRR, but it inserts the appropriate memory management method calls for you at compile-time. You are strongly encouraged to use ARC for new projects. If you use ARC, there is typically no

runtime environment.

At a Glance

need to understand the underlying implementation described in this document, although it may in some situations be helpful. For more about ARC, see *Transitioning to ARC Release Notes*.

Objective-C provides two methods of application memory management.

아래에 잘못된 메모리 관리는 아래의 두 가지의 문제를 일으킵니다.

Freeing or overwriting data that is still in use

See Memory Management Policy.

See Practical Memory Management.

단계들이 있습니다.

Objective-C는 두 가지의 메모리 관리에 대한 방법을 제공합니다.

오토레퍼런스카운팅(ARC)는 위의 MRR과 똑같은 레퍼런스 카운팅을 사용합니다. 하지만 ARC는 컴파일(compile-time)시 적절한 메모리 관리를 위한 메소드를 사용하게 됩니다. 여러분의 프로젝트엔 ARC를 사용하는 것을 권장합니다. 만약 여러분이 ARC를 사용한다면 비록 몇몇 상황에서 유용할지라도 이 문서 아래에 기술된 구현방법을 이해할 필요는 없습 니다.

1. In the method described in this guide, referred to as "manual retain-release" or MRR, you explicitly manage memory by keeping track of objects

이 문서에서 "manual retain-release"라 불리는 이른바 MRR에서 여러분이 생성한 객체들을 계속 주시하여 메모리를 명시적으로 관리하게 됩니다. 이는 레퍼런스 카운팅

you own. This is implemented using a model, known as reference counting, that the Foundation class NSObject provides in conjunction with the

There are two main kinds of problem that result from incorrect memory management:

아직 사용 되고 있는 데이타를 해제하거나 덧씌울 때 This causes memory corruption, and typically results in your application crashing, or worse, corrupted user data. 이는 메모리 충돌을 일으키며 일반적으로 여러분의 app이 종료되거나 잘못동작하거나 잘못된 user data를 사용하게 되는 결과를 나타낼 수 있습니다.

A memory leak is where allocated memory is not freed, even though it is never used again. Leaks cause your application to use ever-increasing

consider memory management in terms of the implementation details rather than in terms of your actual goals. Instead, you should think of

빠지거나 app이 강제적으로 종료될 수 있습니다.

Not freeing data that is no longer in use causes memory leaks

더 이상 사용되지 않는 메모리를 해제하지 않으면 메모리 누수가 발생합니다.

memory management from the perspective of object ownership and object graphs. 여러분은 메모리 관리를 객체의 오너쉽(소유권), object graphs관점으로 생각하는게 좋습니다.

Cocoa는 객체가 메소드에 의해 반환되어 자기의 소유가 되었을 때를 나타내는 간단한 네이밍 규칙을 사용합니다.

Cocoa uses a straightforward naming convention to indicate when you own an object returned by a method.

amounts of memory, which in turn may result in poor system performance or your application being terminated.

Although the basic policy is straightforward, there are some practical steps you can take to make managing memory easier, and to help to ensure your program remains reliable and robust while at the same time minimizing its resource requirements.

Autorelease pool(오토릴리스풀)은 여러분이 "deferred(지연된)" release 메시지를 보낼 수 있는 매커니즘을 제공합니다. 이것은 객체의 오너쉽(소유권)을 포기하고 할당 해제 되지 않기 위한 상황에서 유용하게 사용됩니다. See Using Autorelease Pool Blocks.

Autorelease pool blocks provide a mechanism whereby you can send an object a "deferred" release message. This is useful in situations where you want to relinquish ownership of an object, but want to avoid the possibility of it being deallocated immediately (such as when you return an

• Many of the tools and techniques are described in Technical Note TN2239, iOS Debugging Magic, in particular the use of NSZombie to help find over-released object. • You can use Instruments to track reference counting events and look for memory leaks. See Collecting Data on Your App.

The memory management model is based on object ownership. Any object may have one or more owners. As long as an object has at least one

You create an object using a method whose name begins with "alloc", "new", "copy", or "mutableCopy" (for example, alloc, newObject, or mutableCopy). 여러분은 객체를 생성할 때 "alloc", "new", "copy", or "mutableCopy" 등의 메소드를 이용합니다. 이런 메소드는 생성한 쪽에 리시버 객체를 소유하도록 합니다.

You can take ownership of an object using retain

retain 메소드를 사용해 객체의 소유권을 가질 수 있습니다.

object and when you do not, Cocoa sets the following policy:

유하지 않을 때를 확실히 하기 위해선, Cocoa에선 아래와 같은 규칙들을 정해 놓았습니다.

retain메소드를 사용할 상황은 두 가지로 볼 수 있습니다. (1) 프로퍼티 값을 저장할 객체의 소유권을 갖기 위해 accessor메소드 또는 init메소드 사용할 때; 그리고 (2) 다른 작 업의 부작용으로 객체가 무효화되지 않기 하기 위해서 • When you no longer need it, you must relinquish ownership of an object you own

You relinquish ownership of an object by sending it a release message or an autorelease message. In Cocoa terminology, relinquishing

A received object is normally guaranteed to remain valid within the method it was received in, and that method may also safely return the object to its invoker. You use retain in two situations: (1) In the implementation of an accessor method or an init method, to take ownership of an object you want to store as a property value; and (2) To prevent an object from being invalidated as a side-effect of some other operation (as explained in

This is just corollary of the previous policy rules, stated explicitly.

// ...

NSString *string = [[[NSString alloc] initWithFormat:0"%0 %0", self.firstName, self.lastName] autorelease]; return string; You own the string returned by alloc. To abide by the memory management rules, you must relinquish ownership of the string before you lose the reference to it. If you use release, however, the string will be deallocated before it is returned (and the method would return an invalid object). Using autorelease, you signify that you want to relinquish ownership, but you allow the caller of the method to use the returned string before it is deallocated. 메모리 관리 규칙을 준수하기 위해선, 객체를 참조한 것을 잃기 전에 그 객체의 소유권을 포기해야 합니다. 그러나 release를 사용하면 문자객체가 반환 되기 전에 할당 해제가 될것

You could also implement the fullName method like this:

By way of contrast, the following implementation is wrong:

NSString *string = [[NSString alloc] initWithFormat:0"%0 %0",

therefore has no reason to release the returned string, and it will thus be leaked.

pattern is to use an NSError object that contains information about an error if one occurs, as illustrated by

object, so you do not own it. There is therefore no need to release it, as illustrated in this example:

initWithContentsOfURL:options:error: (NSData) and initWithContentsOfFile:encoding:error: (NSString).

Following the basic rules, you don't own the string returned by stringWithFormat:, so you can safely return the string from the method.

기본적인 규칙에 따르면, stringWithFormat:메소드는 반환되는 문자 객체를 소유하지 않습니다. 그리하여 안전하게 해당 객체를 반환할 수 있습니다.

self.firstName, self.lastName];

NSString *string = [[NSString alloc] initWithContentsOfFile:fileName encoding:NSUTF8StringEncoding error:&error];

NSString *fileName = <#Get a file name#>;

NSError *error;

if (string == nil) {

// ...

@end

// ...

@end

주세요

- (void)dealloc

[firstName release];

[lastName release];

[string release];

// Deal with error...

@interface Person : NSObject

@property (retain) NSString *firstName;

@property (retain) NSString *lastName;

@property (assign, readonly) NSString *fullName;

The NSObject class defines a method, dealloc, that is invoked automatically when an object has no owners and its memory is reclaimed—in

[super dealloc];

naming conventions for Cocoa and Core Foundation, however, are different. In particular, Core Foundation's Create Rule (see The Create Rule) does not apply to methods that return Objective-C objects. For example, in the following code fragment, you are *not* responsible for relinquishing

Good Practices Prevent Memory-Related Problems

메모리 누수는 더 이상 사용하지 않는 메모리를 해제하지 않았을 때 일어납니다. 이러한 메모리 누수는 여러분 app의 메모리 사용량을 계속 증가시키게 되고 그 결과 시스템 성능이 나 Thinking about memory management from the perspective of reference counting, however, is frequently counterproductive, because you tend to

Use Analysis Tools to Debug Memory Problems

Memory Management Policy

provides some examples of correct usage.

You own any object you create

A Simple Example

객체를 생성할 때 생성한 쪽이 그 객체를 소유한다

메모리관리 기본정책은 간단하지만, 메모리 관리를 쉽게하고 시스템 리소스 요구를 최소한으로 하면서 여러분의 프로그램의 신뢰성과 견고성을 보장하는데 유용한 몇 가지의 실용적인

object from a method). There are occasions when you might use your own autorelease pool blocks.

To identify problems with your code at compile time, you can use the Clang Static Analyzer that is built into Xcode. If memory management problems do nevertheless arise, there are other tools and techniques you can use to identify and diagnose the issues.

레퍼런스 카운트 환경에서 메모리 관리를 사용하는 기본적인 모델은 NSObject에 정의된 메소드를 조합하여 제공됩니다. 또한 NSObject클래스는 객체가 메모리해제될 때 자동으 로 호출되는 dealloc 메소드를 정의 합니다. Basic Memory Management Rules

owner, it continues to exist. If an object has no owners, the runtime system destroys it automatically. To make sure it is clear when you own an

메모리 관리 모델은 객체의 소유권개념을 기본으로 합니다. 모든 객체는 하나 이상의 owner(소유자)를 가질 수 있습니다. 객체가 하나 이상의 owner를 가질 수록 그 객체는 할당 해 제되지 않고 계속 존재할 가능성이 높습니다. 만약 어떤 객체에 owner가 존재하지 안흐면 runtime system은 자동으로 그 객체를 파기합니다. 여러분이 언제 객체를 소유하거나 소

The basic model used for memory management in a reference-counted environment is provided by a combination of methods defined in the

when an object is deallocated. This article describes all the basic rules you need to know to manage memory correctly in a Cocoa program, and

NSObject protocol and a standard method naming convention. The NSObject class also defines a method, dealloc, that is invoked automatically

Avoid Causing Deallocation of Objects You're Using).

해당 객체가 더 이상 필요 없을 때, 여러분은 반드시 해당 객체의 소유를 포기해야합니다.

You must not relinquish ownership of an object you do not own

To illustrate the policy, consider the following code fragment:

Person *aPerson = [[Person alloc] init];

NSString *name = aPerson.fullName;

[aPerson release];

rather than autorelease.

구현할 수 있습니다.

해야 할 것입니다.

return string;

하지만 아래와 같은 구현은 옳지 안습니다!!

- (NSString *)fullName {

return string;

- (NSString *)fullName {

ownership of an object is therefore typically referred to as "releasing" an object.

소유하지 않을 객체에 소유권을 포기하는 메소드를 호출해선 안됩니다.

release 또는 autorelease 메소드를 사용해 객체의 소유권을 포기할 수 있습니다. Cocoa 용어로 객체의 소유권을 포기하는 것은 객체를 "releasing"한다 라고도 부릅니다.

// ...

The Person object is created using the alloc method, so it is subsequently sent a release message when it is no longer needed. The person's

You use autorelease when you need to send a deferred release message—typically when returning an object from a method. For example, you

autorelease는 주로 메소드로 객체를 반환해야 하는 때에 release메시지를 좀 더 지연시켜서 나중에 되게 하기 위하여 사용합니다. 예를 들어 fullName메소드를 아래와 같이

name is not retrieved using any of the owning methods, so it is not sent a release message. Notice, though, that the example uses release

Person 객체는 alloc메소드를 통해 생성되었고, 더 이상 필요없을 때 release메시지(메소드)를 보내야합니다. name 프로퍼티는 객체를 소유하지 않는 메소드를 통해 만들어 지 므로 release메시지를 보낼 필요가 없습니다.

Use autorelease to Send a Deferred release

could implement the fullName method like this:

- (NSString *)fullName { NSString *string = [NSString stringWithFormat:0"%0 %0",

self.firstName, self.lastName];

입니다(그리고 메소드는 유효하지않은 객체를 반환할 것입니다). autorelease를 사용하여 객체의 소유권을 포기하고 메소드를 호출한 쪽에 할당 해제되기 전에 유효한 객체를 반환

명명 규칙에 따르면 fullName메소드를 호출하는 쪽이 반환되는 객체의 소유자가 된다는 것을 이라는 fullName이름을 보고 알지 못할 것입니다. 호출한 쪽은 반환받은 객체를 release를 해야 한다는 사실을 모를 것입니다. 그렇게 되면 메무리 누수가 발생할 것입니다. You Don't Own Objects Returned by Reference

Some methods in Cocoa specify that an object is returned by reference (that is, they take an argument of type ClassName ** or id *). A common

In these cases, the same rules apply as have already been described. When you invoke any of these methods, you do not create the NSError

According to the naming convention, there is nothing to denote that the caller of the fullName method owns the returned string. The caller

Cocoa terminology it is "freed" or "deallocated.". The role of the dealloc method is to free the object's own memory, and to dispose of any resources it holds, including ownership of any object instance variables. NSObject에서는 어떤 객체에 소유자가 존재하지 않고 그 객체가 존재하는 메모리가 재생(메모리에 다른 정보가 씌워질 때?)될 때 dealloc메소드를 자동으로 호출합니다. dealloc메소드는 그 객체가 소유한 메모리를 해제하고 그 객체가 소유한 모든 인스턴스 변수의 소유권을 포함한 여러 자원들을 처리하는 기능을 합니다.

Implement dealloc to Relinquish Ownership of Objects

The following example illustrates how you might implement a dealloc method for a Person class:

@implementation Person

시스템 리소스의 관리를 객체의 lifetime(생명주기)와 연결지어 구성하지 않는 것이 좋습니다. 자세한 내용은 Don't Use dealloc to Manage Scarce Resources을 참고해

Important: Never invoke another object's dealloc method directly.

절대로!! 객체의 dealloc메소드를 직접 호출해선 안됩니다!!

ownership of myInstance:

MyClass *myInstance = [MyClass createInstance];

When an application terminates, objects may not be sent a dealloc message. Because the process's memory is automatically cleared on exit, it is more efficient simply to allow the operating system to clean up resources than to invoke all the memory management methods. App이 종료될 때, 객체들은 dealloc메시지를 보내지 않을 수도 있습니다. 종료 시 프로세스의 메모리가 자동으로 비워지게 되므로 메모리 관리와 관련된 메소드를 호출하기 보 다는 operating system이 리소스들을 비우게 하는 편이 좀 더 효과적입니다. Core Foundation Uses Similar but Different Rules There are similar memory management rules for Core Foundation objects (see *Memory Management Programming Guide for Core Foundation*). The

You must invoke the superclass's implementation at the end of your implementation.

반드시 여러분의 오버라이드한 dealloc메소드구현 마지막에 부모클래스의 dealloc메소드를 호출해야 합니다.

You should not tie management of system resources to object lifetimes; see Don't Use dealloc to Manage Scarce Resources.