OBJ-C MEMORY MANAGEMENT

Memory management is one of the most important process in any programming language. It is the process by which the memory of objects are allocated when they are required and deallocated when they are no longer required.

Managing object memory is a matter of performance; if an application doesn't free unneeded objects, its memory footprint grows and performance suffers.

Objective-C Memory management techniques can be broadly classified into two types.

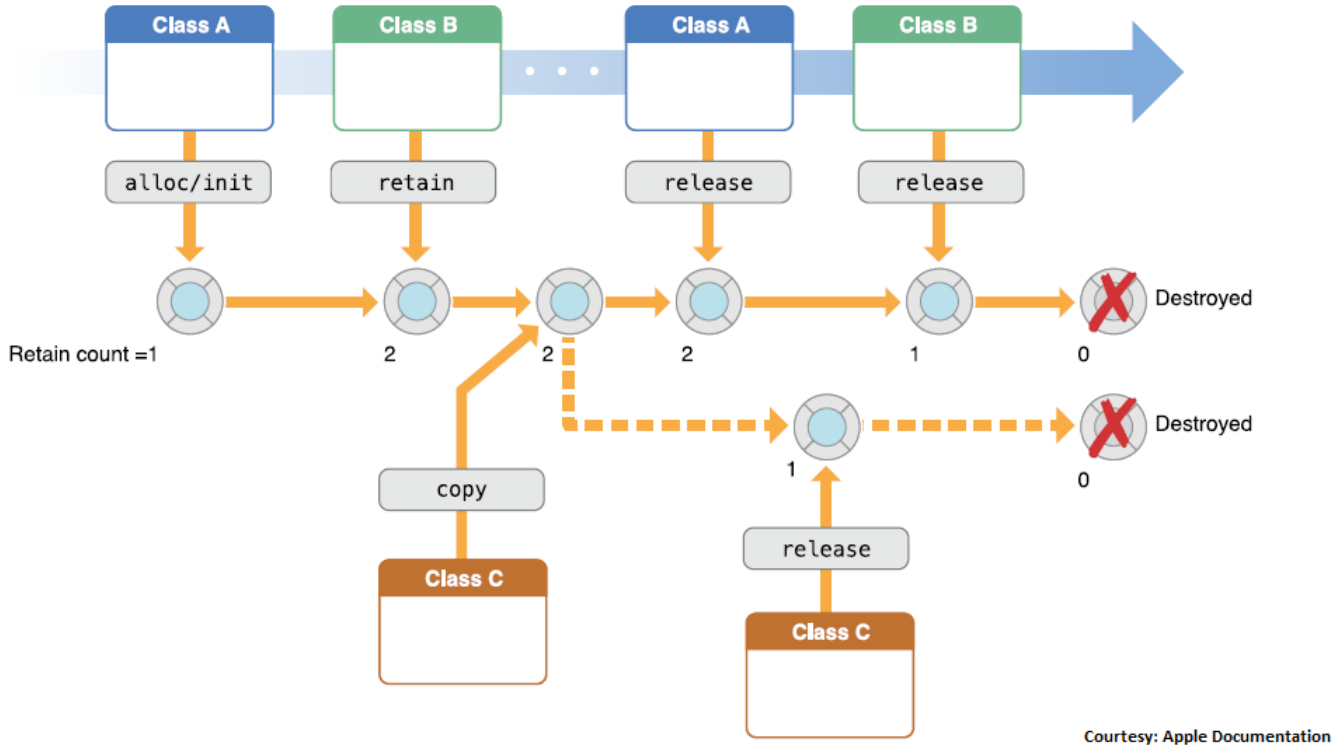
- "Manual Retain-Release" or MRR
- "Automatic Reference Counting" or ARC

"Manual Retain-Release" or MRR

In MRR, we explicitly manage memory by keeping track of objects on our own. This is implemented using a model, known as reference counting, that the Foundation class NSObject provides in conjunction with the runtime environment.

The only difference between MRR and ARC is that the retain and release is handled by us manually in former while its automatically taken care of in the latter.

The following figure represents an example of how memory management work in Objective-C.



The memory life cycle of the Class A object is shown in the above figure. As you can see, the retain count is shown below the object, when the retain count of an object becomes 0, the object is freed completely and its memory is deallocated for other objects to use.

Class A object is first created using alloc/init method available in NSObject. Now, the retain count becomes 1.

Now, class B retains the Class A's Object and the retain count of Class A's object becomes 2.

Then, Class C makes a copy of the object. Now, it is created as another instance of Class A with same values for the instance variables. Here, the retain count is 1 and not the retain count of the

original object. This is represented by the dotted line in the figure.

The copied object is released by Class C using the release method and the retain count becomes 0 and hence the object is destroyed.

In case of the initial Class A Object, the retain count is 2 and it has to be released twice in order for it to be destroyed. This is done by release statements of Class A and Class B which decrements the retain count to 1 and 0, respectively. Finally, the object is destroyed.

MRR Basic Rules

- We own any object we create: We create an object using a method whose name begins with "alloc", "new", "copy", or "mutableCopy"
- We can take ownership of an object using retain: 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. We use retain in two situations:
    - In the implementation of an accessor method or an init method, to take ownership of an object we want to store as a property value.
    - To prevent an object from being invalidated as a side-effect of some other operation.
- When we no longer need it, we must relinquish ownership of an object we own: We relinquish ownership of an object by sending it a release message or an autorelease message. In Cocoa terminology, relinquishing ownership of an object is therefore typically referred to as "releasing" an object.
- You must not relinquish ownership of an object you do not own: This is just corollary of the previous policy rules stated explicitly.

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

@interface SampleClass:NSObject
- (void)sampleMethod;
@end

@implementation SampleClass

- (void)sampleMethod
{
   NSLog(@"Hello, World! \n");
}

- (void)dealloc
{
  NSLog(@"Object deallocated");
  [super dealloc];
}

@end

int main()
{
   /* my first program in Objective-C */
   SampleClass *sampleClass = [[SampleClass alloc]init];
   [sampleClass sampleMethod];
   NSLog(@"Retain Count after initial allocation: %d",
   [sampleClass retainCount]);
   [sampleClass retain];
   NSLog(@"Retain Count after retain: %d", [sampleClass retainCount]);
   [sampleClass release];
   NSLog(@"Retain Count after release: %d", [sampleClass retainCount]);
   [sampleClass release];
   NSLog(@"SampleClass dealloc will be called before this");
   // Should set the object to nil
   sampleClass = nil;
   return 0;
```

```
}
```

When we compile the above program, we will get the following output.

```
2013-09-28 04:39:52.310 demo[8385] Hello, World!
2013-09-28 04:39:52.311 demo[8385] Retain Count after initial allocation: 1
2013-09-28 04:39:52.311 demo[8385] Retain Count after retain: 2
2013-09-28 04:39:52.311 demo[8385] Retain Count after release: 1
2013-09-28 04:39:52.311 demo[8385] Object deallocated
2013-09-28 04:39:52.311 demo[8385] SampleClass dealloc will be called before this
```

"Automatic Reference Counting" or ARC

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 us at compile-time. We are strongly encouraged to use ARC for new projects. If we use ARC, there is typically no 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.

As mentioned above, in ARC, we need not add release and retain methods since that will be taken care by the compiler. Actually, the underlying process of Objective-C is still the same. It uses the retain and release operations internally making it easier for the developer to code without worrying about these operations, which will reduce both the amount of code written and the possibility of memory leaks.

There was another principle called garbage collection, which is used in Mac OS-X along with MRR, but since its deprecation in OS-X Mountain Lion, it has not been discussed along with MRR. Also, iOS objects never had garbage collection feature. And with ARC, there is no use of garbage collection in OS-X too.

Here is a simple ARC example. Note this won't work on online compiler since it does not support ARC.

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

@interface SampleClass:NSObject
- (void)sampleMethod;
@end

@implementation SampleClass

- (void)sampleMethod
{
   NSLog(@"Hello, World! \n");
}

- (void)dealloc
{
  NSLog(@"Object deallocated");
}

@end

int main()
{
   /* my first program in Objective-C */
   @autoreleasepool{
       SampleClass *sampleClass = [[SampleClass alloc]init];
       [sampleClass sampleMethod];
    sampleClass = nil;
   }
   return 0;
}
```

When we compile the above program, we will get the following output.

```
2013-09-28 04:45:47.310 demo[8385] Hello, World!
2013-09-28 04:45:47.311 demo[8385] Object deallocated
```