


CSC 471 / 371 Mobile Application Development for iOS




Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
[@DePaulSWEng](https://twitter.com/DePaulSWEng)




1

Swift Primer, Part 6 Memory Management and ARC



2

Outline






- Memory management
- Automatic reference counting

DePAUL UNIVERSITY

3

Memory Leaks




- Any allocated memory that is not properly returned to the operating system is a *memory leak*
 - Not freeing data that is no longer in use
- Leaks are bad in many ways
 - Consume and lose valuable resources
 - Use ever-increasing amounts of memory
 - Prevent applications from running for a long duration
 - Fragment memory, prevent larger allocations
 - Poor system performance
 - iOS may terminate some apps due to low memory

DePAUL UNIVERSITY

4

Other Memory Related Bugs




- Freeing data that are still in use
- Using data that are already freed
- These will have serious consequences
 - Memory corruption
 - Corrupted user data
 - Application crashing

DePAUL UNIVERSITY

5

Memory Management



- C/C++ style memory management
 - Memory is allocated, tracked and released by programmers.
- Advantages
 - Complete control of the life span of the memory
 - Fast, efficient, less overhead
- Disadvantages
 - Memory leaks or other related bugs often occur
 - Memory related bugs are difficult to track down
 - Expensive for testing, debugging, and QA

DePAUL UNIVERSITY

6

Garbage Collection

- Java-style garbage collected environment
- Release is a no-op
- Uses a mark and sweep process
 - Time consuming
 - Slow down normal operations
 - May stop all threads as the garbage collection happens
 - Unpredictable timing
 - Timer or resource based trigger
- *iOS doesn't support garbage collection!*

DEPAUL UNIVERSITY

7

Automated Reference Counting

- iOS supports *Automated Reference Counting* (ARC)
 - Frees memory when it is no longer needed
 - Will not free memory as long as at least one active reference to that memory still exists.
- Every object has a *retain* count
 - How many references are pointed at the object.
 - If retain count > 0
 - The object is alive, and will not be freed
 - If retain count == 0
 - The object is not reference by any other objects, i.e., it is garbage
 - The memory allocated for the object is released.

DEPAUL UNIVERSITY

8

How Does ARC Work?

- A simple class
 - The `deinit` method is for *deinitializing* before an object is released.
 - Non-memory related cleaning-up. Sometimes needed.

```
class Person {
    let name: String
    init(name: String) {
        self.name = name
        print("\(name) is being initialized")
    }
    deinit {
        print("\(name) is being deinitialized")
    }
}
```

Invoked
before an
object is
released

DEPAUL UNIVERSITY

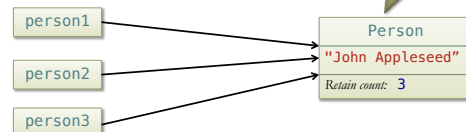
9

How Does ARC Work?

```
var person1: Person?
person1 = Person(name: "John Appleseed")

var person2 = person1
var person3 = person1
```

ARC automatically
manages the retain
count of all objects



DEPAUL UNIVERSITY

10

How Does ARC Work?

```
person1 = nil
person2 = nil
person3 = nil
```

No reference to the object.
Will be deinitialized and
freed (at a time determined
by the system)

```
person1
person2
person3
```

```
Person
"John Appleseed"
Retain count: 0
```

DEPAUL UNIVERSITY

11

Reference Cycles

- It is possible to have reference cycles, which are **problematic**
- Consider the following classes

```
class Person {
    let name: String
    init(name: String) {
        self.name = name
    }
    var apartment: Apartment?
}

class Apartment {
    let number: Int
    init(number: Int) {
        self.number = number
    }
    var tenant: Person?
}
```

DEPAUL UNIVERSITY

12

Reference Cycles

```
var john: Person?
var number73: Apartment?

john = Person(name: "John Appleseed")
number73 = Apartment(number: 73)
```

DEPAUL UNIVERSITY 13

Reference Cycles

```
john!.apartment = number73
number73!.tenant = john
```

DEPAUL UNIVERSITY 14

Reference Cycles

```
john = nil
number73 = nil
```

DEPAUL UNIVERSITY 15

Strong and Weak References

- By default, references are *strong*
 - Increment the retain count of the referenced object
 - An object referenced by a strong reference can never be freed
- Weak* references do not prevent referenced object to be freed
 - Do not increment the retain count of the referenced object
 - Can be used to break the strong reference cycles
 - Referenced objects may disappear! Become *nil*
- An object must be referenced by *at least one strong reference* to stay alive

DEPAUL UNIVERSITY 16

Automatic Reference Counting

- The system automatically manages when to retain and when to release objects.
 - Minimum overhead, highly efficient
- Programmers don't have to explicitly deal with memory allocation and deallocation.
 - No need to call any method to manage memory
- Programmers need to determine whether a reference should be *strong* or *weak*.
- Value types (struct, enum, etc.) are not managed by ARC
 - Deallocated when exit the scope

DEPAUL UNIVERSITY 17

Reference Cycle Revisited

- Break the strong reference cycle
 - Strong references should never form a cycle**
- Declare one of the reference in the cycle to be weak

```
class Apartment {
    let number: Int
    init(number: Int) {
        self.number = number
    }
    weak var tenant: Person?
}
```

Any reference type variable can be declared weak.

DEPAUL UNIVERSITY 18

Reference Cycle Revisited

```
var john: Person?
var number73: Apartment?
john = Person(name: "John Appleseed")
number73 = Apartment(number: 73)
john!.apartment = number73
number73!.tenant = john
```

No reference cycle.

DEPAUL UNIVERSITY

Reference Cycle Revisited

```
john = nil
```

No strong reference to the object. It can be freed. Weak references to the object are set to nil.

DEPAUL UNIVERSITY

Reference Cycle Revisited

```
john = nil
number73 = nil
```

No strong reference to the object. It can be freed.

DEPAUL UNIVERSITY

Strong or Weak References

Principles:

1. **No strong reference cycle**
2. Use strong reference if the referenced object should not be released while the referencing object is alive.
3. At least one strong reference is needed to keep an object alive.

DEPAUL UNIVERSITY

Strong or Weak References

- UI view hierarchy created in IB maintains strong references from top down.
 - For the root view to all its subviews
- The reference from a view controller to its associated view is strong
 - Views will not disappear by themselves

DEPAUL UNIVERSITY

Strong or Weak References

- Outlets created in IB can be strong or weak
- Weak outlet reference is recommended
 - Allow the view hierarchy to be freed more easily if necessary
- Strong outlet reference creates no cycle
 - To free the view hierarchy, all strong outlet references must be set to nil too.

Set this reference to nil

Strong outlet references must be nil too

DEPAUL UNIVERSITY

Next ...

- Multi-threads
- Grand Central Dispatch

❖ iOS is a trademark of Apple Inc.

DEPAUL UNIVERSITY

25

