Memory Management

-Devendranath

For Offline / Online Training, reach me@ iPhoneDev1990@gmail.com

Swift uses a system of memory management called **Automatic Reference Counting** (ARC). ARC keeps track of how many references are there to an object in your code. When you create a new reference to something the reference count for that object increases

Memory management functions and its usage are handled in Swift language through **Automatic reference counting** (ARC). ARC is used to **initialize** and **deinitialize** the system resources thereby releasing memory spaces used by the class instances when the instances are no longer needed.

ARC keeps track of information about the relationships between our code instances to manage the memory resources effectively.

- ARC allocates a chunk of memory to store the information each and every time when a new class instance is created by init().
- Information about the instance type and its values are stored in memory.
- When the class instance is no longer needed it automatically frees the memory space by deinit() for further class instance storage and retrieval.
- ARC keeps in track of currently referring class instances properties, constants and variables so that deinit() is applied only to those unused instances.
- ARC maintains a 'strong reference' to those class instance property, constants and variables to restrict deallocation when the class instance is currently in use.

var aAeroplane = Aeroplane() 1

. . . .

. . . .

Use aeroplane object

. . . .

. . . .

ARC releases the aAeroplane object, if it is no longer needed. 0

init() increases the reference count by 1.
deinin() decreases the reference count by 1.
ARC releases the object's memory once the reference count become **0**.

Note: All variables declared in class are by default strong references.

Strong, weak and unowned

Strong, **weak**, and **unowned** are keywords that describe the nature of a reference in the ARC paradigm.

All references are **strong** references by **default** unless otherwise specified. **Most of the time**, **this is the right thing to do**. **Strong** is always implied when you declare a variable or constant. You don’t need to type strong in Swift.

ARC doesn’t free up the memory being used by a class instance until all strong references to that instance are broken. How are strong references broken?

Strong References can be broken

- When a variable that references an instance of something is set to nil
- When a parent variable that holds a reference to a child class instance is to nil it will break the reference to the parent and the child
- When a variable or constant goes out of scope – for example, if something gets initialized inside a control-flow code segment like an if/else or inside a for loop, when execution moves past that code segment, the reference is broken and the memory is freed by ARC

Weak and unowned references

Weak and **unowned** references come into the picture when we start talking about the **relationships** that emerge between class instances.

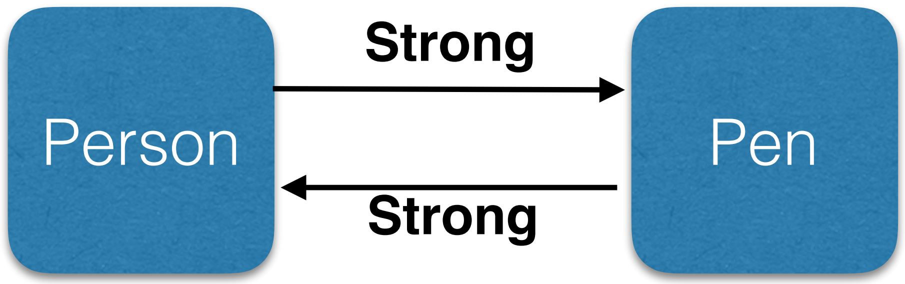
Relationships between classes are core to the Object-Oriented paradigm. Whether you plan them or not, these relationships exist, and they affect the ARC memory management model.

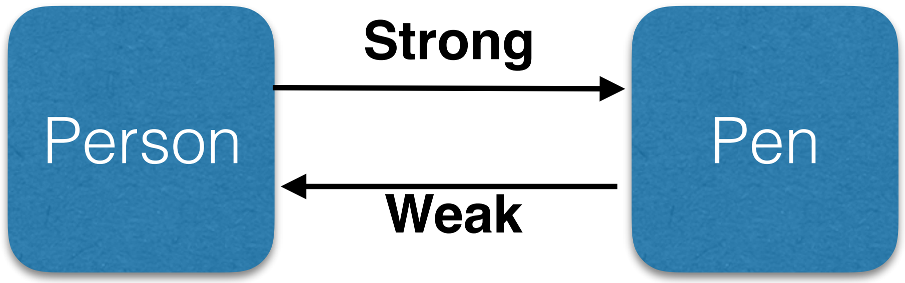
When two instances are optionally related to one another, make sure that one of those instances holds a **weak** reference to the other.

When two instances are related in such a way that one of the instances can’t exist without the other, the instance with the mandatory dependency needs to hold an **unowned** reference to the other instance.

Case to Handle by Developer

Swift ARC takes care of most of the cases except Retail Cycle case.





Retain Cycle: Retain cycle occurs in languages where the memory management is based on Reference counting.

When Two objects are pointing to each other strongly, ARC can't find a way to release them. In such situations Memory leaks happens.

For Offline/ Online Training, reach me@ iPhoneDev1990@gmail.com

Code Snippet

```
class MemoryManagement
{
    func playWithreferences()
    {
        var aHuman: Human? = Human(aName: "DevReddy")
        aHuman?.aHeart = Heart(pulse: 72)
        aHuman = nil;
    }
}

class Human
{
    var name: String
    var aHeart: Heart!

    init(aName: String) {
        print("Creating a Human Object")
        self.name = aName;
    }
    deinit {
        print("Human object released");
    }
}
//OUTPUT
Creating a Human Object
Pulse object is created
Human object released
Releasing pulse object
```

Retain Cycle Occurrence

```
class MemoryManagement
{
    func playWithreferences()  {
        var aHuman: Human? = Human(aName: "DevReddy")
        var aHeart: Heart? = Heart(pulse: 72)

        aHuman?.aHeart = aHeart;
        aHeart?.aOwner = aHuman;
        aHuman = nil;
       aHeart = nil;
    }
}

class Human
{
    var name: String
    var aHeart: Heart!

    init(aName: String) {
        print("Creating a Human Object")
        self.name = aName;
    }
    deinit {
        print("Human object released");
    }
}

class Heart
{
    var pulse: Int!
    var aOwner: Human!

    init(pulse: Int)  {
        print("Pulse object is created");
        self.pulse = pulse;
    }
    deinit  {
        print("Releasing pulse object")
    }
}
```

Output:

```
Creating a Human Object
Pulse object is created
```

Retain Cycle Occurrence

```
class MemoryManagement
{
    func playWithreferences()  {
        var aHuman: Human? = Human(aName: "DevReddy")
        var aHeart: Heart? = Heart(pulse: 72)

        aHuman?.aHeart = aHeart;
        aHeart?.aOwner = aHuman;
        aHuman = nil;
       aHeart = nil;
    }
}

class Human
{
    var name: String
    var aHeart: Heart!

    init(aName: String) {
        print("Creating a Human Object")
        self.name = aName;
    }
    deinit {
        print("Human object released");
    }
}

class Heart
{
    var pulse: Int!
    weak var aOwner: Human!

    init(pulse: Int)  {
        print("Pulse object is created");
        self.pulse = pulse;
    }
    deinit  {
        print("Releasing pulse object")
    }
}
```

Output:

```
Creating a Human Object
Pulse object is created
```

Output:

```
Creating a Human Object
Pulse object is created
Human object released
Releasing pulse object
```

Thank You