Automatic Reference Counting

Swift uses *Automatic Reference Counting (ARC)* to track and manage your app's memory usage. In most cases, this means that memory management "just works" in Swift, and you don't need to think about memory management yourself. ARC automatically frees up the memory used by class instances when those instances are no longer needed.

Resolving Strong Reference Cycles Between Class Instances

Swift provides two ways to resolve strong reference cycles when you work with properties of class type: weak references and unowned references.

Weak and unowned references enable one instance in a reference cycle to refer to the other instance *without* keeping a strong hold on it. The instances can then refer to each other without creating a strong reference cycle.

Use a weak reference when the other instance has a shorter lifetime—that is, when the other instance can be deallocated first. In the `Apartment` example below, it's appropriate for an apartment to be able to have no tenant at some point in its lifetime, and so a weak reference is an appropriate way to break the reference cycle in this case. In contrast, use an unowned reference when the other instance has the same lifetime or a longer lifetime.

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

class Apartment {
```

```
    let unit: String
    init(unit: String) { self.unit = unit }
    weak var tenant: Person?
    deinit { print("Apartment \(unit) is being
deinitialized") }
}
```

A weak reference is a reference that doesn't keep a strong hold on the instance it refers to, and so doesn't stop ARC from disposing of the referenced instance. Like a weak reference, an unowned reference doesn't keep a strong hold on the instance it refers to. Unlike a weak reference, however, an unowned reference is used when the other instance has the same lifetime or a longer lifetime. You indicate an unowned reference by placing the `unowned` keyword before a property or variable declaration.

When to use : —> Use a [weak] reference whenever it is valid for that reference to become nil at some point during its lifetime. Conversely, use an [unowned] reference when you know that the reference will never be nil once it has been set during initialization.

It's best practice to always use weak combined with self inside closures to avoid retain cycles.

Define a capture in a closure as an unowned reference when the closure and the instance it captures will always refer to each other, and will always be deallocated at the same time.

Weak references are always of an optional type, and automatically become `nil` when the instance they reference is deallocated. This enables you to check for their existence within the closure's body.

In this data model, a customer may or may not have a credit card, but a credit card will *always* be associated with a customer. Because a credit card will always have a customer, you define its `customer` property as an unowned reference, to avoid a strong reference cycle:

```
class Customer {
    let name: String
    var card: CreditCard?
```

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

class CreditCard {
    let number: UInt64
    unowned let customer: Customer
    init(number: UInt64, customer: Customer) {
        self.number = number
        self.customer = customer
    }
    deinit { print("Card #\(number) is being
deinitialized") }
}
```