Memory Management

Dr.-Ing. Thomas Springer
M.Sc. Martin Weißbach
https://www.rn.inf.tu-dresden.de/martin/iOS-Programming-17/
Memory-Management.zip
Reference & Value Types

**Reference Types**

- not copied when assigned to a variable or constant
- not copied when passed to a function as a parameter
- rather, a reference is used (comparable to pointer semantics)

**Value Types**

- copied when assigned to a variable or constant
- copied when passed to a function as a parameter
Reference Counting
• Reference Counting

  • Idea: Count references of objects to other objects to determine if an object is still required
  • Clear memory if object is not referenced anymore
  • Applies only to reference types, e.g. Classes & Closures
Reference Counting

A
Field 1
rc = 1
Field 2

B
Field 1
rc = 3
Field 2

C
Field 1
rc = 1
Field 2

obj
### Properties

```swift
{class,struct,enum} Example {
    var prop1: OtherExample
    let prop2: OtherExample
}
```

### Variables & Constants

```swift
let test = Example()
var test = Example()
{var, let} third = test
```

### Captured inside a closure’s body

```swift
var captured = Example()
prop.handler = {
    () -> Void in
    captured.prop1 = Example()
}
```
Retain Cycles
Retain Cycles: Classes

class Apartment {
  var street, zipCode, city
  var tenant: Tenant?
}

class Person {
  let name, age
}

class Tenant: Person {
  var address: Apartment?
}

TwoRoom: Apartment
street: SwiftRoad 2
zipCode: 12345
city: SwiftCity
tenant: Levy

Levy: Tenant
address: TwoRoom

strong references
Weak & Unowned References

• do not keep a strong reference to the referred instance

  Weak References
  • referenced values can become nil
  • weak references must be variables and optional types

  Unowned References
  • referenced values cannot become nil
  • can be accessed directly → runtime error if referenced instance was already deallocated
class Apartment {
    var street, zipCode, city
    weak var tenant: Tenant?
}

class Person {
    let name, age
}

class Tenant: Person {
    var address: Apartment?
}
Retain Cycles: Closures

Asynchronous Task

View Controller

Completion Handler

owns / performs

owns / provides

self-reference

implments / self-reference
class ViewController {
    //...
    let task: AsyncTask
    //...
}

class AsyncTask {
    //...
    var completionHandler: ( () -> Void )?
    //...
}

task.completionHandler = { () -> Void in
    //...
    self.result = 42
    //...
Capture List

- declared inside closure object
- define the set of captured variables that are only held by unowned references by the closure

```swift
var closure: (String) -> Void = {
    [unowned self, weak delegate = self.delegate] (name: String) -> Void in
    // closure body
}

var closure: Void -> Void = {
    [unowned self, weak delegate = self.delegate] in
    // closure body
}
```
class ViewController {
    //...
    let task: AsyncTask
    //...
}

task.completionHandler = { [weak self] () -> Void in
    //...
    self?.result = 42
    //...
}