Технічні науки

УДК 004.852

Петрішенко Сергій Олександрович
студент
Національний технічний університет України
«Київський політехнічний інститут»
Петришенко Сергей Александрович
студент
Национальный технический университет Украины
«Киевский политехнический институт»
Petrishenko S.
student
National Technical University of Ukraine “Kyiv Polytechnic Institute”

УПРАВЛІННЯ ПАМ’ЯТТЮ В IOS ЗАСТОСУВАННЯХ
УПРАВЛЕНИЕ ПАМЯТЬЮ В IOS ПРИЛОЖЕНИЯХ
MEMORY MANAGEMENT IN IOS APPLICATIONS

Анотація: Досліджено основні особливості управління пам’яттю в мобільних застосуваннях для операційної системи iOS на основі мови програмування Swift.

Ключові слова: екземпляр класу, сильне посилання, слабке посилання, невизначене посилання, опціональний тип, ініціалізатор.

Аннотация: Исследованы основные особенности управления памятью в мобильных приложениях для операционной системы iOS на основе языка программирования Swift.

Ключевые слова: экземпляр класса, сильная ссылка, слабая ссылка, неопределенная ссылка, опциональный тип, инициализатор.

Summary: Were investigated main features of memory management in iOS applications using Swift.

Key words: class instance, strong reference, weak reference, unowned reference, optional type, initializer.

Як і в мові Objective-C, в мові Swift застосовується підрахунок посилань в якості основного механізму для відслідковування моментів, коли оперативна пам'ять, яка була динамічно виділена, більше не використовується і може бути звільнення для інших ресурсів. Протягом багатьох років (та й тепер) в мові Objective-C як і раніше застосовується ручний підрахунок посилань. В результаті нетривалого експериментування зі збирачем "сміття" в Objective-C для автоматичного управління пам'яттю, в 2011 році компанія Apple оголосила про розробку механізму ARC (Automatic Reference Counting). У ньому застосовується той же самий підхід, що і при ручному підрахунку посилань, але тільки більш суворим і детермінованим чином.

Принцип дії підрахунку посилань

ARC (automatic reference counting) використовується для відстеження та управління пам'яттю мобільного застосування. ARC автоматично звільняє пам'ять, яка використовувалася екземпляром класу, коли ці екземпляри більше не потрібні. Кожен раз, коли створюється екземпляр класу, ARC виділяє шматок пам'яті для зберігання інформації цього екземпляра. Цей шматок пам'яті містить інформацію про тип екземпляра, про його значення і про будь-які властивості, пов'язаних з ним. Додатково, коли екземпляр більше не потрібен, ARC звільняє пам'ять, використану під цей екземпляр, і направляє цю пам'ять туди, де вона потрібна. Це свого роду гарантія того, що непотрібні екземпляри не будуть займати пам'ять.

Однак, якщо ARC звільнить пам'ять використовуваного екземпляра, то доступ до властивостей або методів цього екземпляра буде неможливий.

Якщо спробувати отримати доступ до цього екземпляра, то додаток швидше за все видасть помилку і зупинить свою роботу. Для того, щоб потрібний екземпляр не пропав, ARC веде облік кількості властивостей, констант, змінних, які посилаються на кожен екземпляр класу. ARC не звільнить екземпляр, якщо є хоча б одне активне посилання. Для того щоб це було можливо, кожен раз як привласнюється екземпляр властивості, константі або змінній створюється strong reference (сильний зв'язок) з цим екземпляром. Такий зв'язок називається "сильним", так як він міцно тримається за цей екземпляр і не дозволяє йому звільниться до тих пір, поки залишаються сильні зв'язки.

Наприклад, нехай є клас Person, який визначає константну властивість name:

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

Клас Person має ініціалізатор, який встановлює властивість name екземпляра і виводить повідомлення для відображення того, що йде ініціалізація. Так само клас Person має деініціалізатор, який виводить повідомлення, коли екземпляр класу звільняється.

Наступний приклад код визначає три змінні класу Person?, який використовується для установки декількох посилань до нового екземпляру Person в наступних шматках коду. Так як ці змінні опціонального типу Person?, а не Person, вони автоматично ініціалізуються зі значенням nil, і не мають жодних посилань на екземпляр Person:

```
var reference1: Person?
```

```
var reference2: Person?
var reference3: Person?
```

Тепер можна створити екземпляр класу Person і привласнити його однією з цих трьох змінних:

```
reference1 = Person(name: "John Appleseed")
// Prints "John Appleseed is being initialized"
```

Повідомлення "John Appleseed is being initialized" виводиться під час того, як викликається ініціалізатор класу Person. Це підтверджує той факт, що відбулася ініціалізація.

Так як новий екземпляр класу Person було присвоєно змінній reference1, значить тепер існує сильне посилання між reference1 і новим екземпляром класу Person. Тепер у цього екземпляра є як мінімум одне сильна посилання, значить ARC тримає під Person пам'ять і не звільняє її. Якщо надати іншим змінним той же екземпляр Person, то додасться два сильних посилання до цього екземпляра:

```
reference2 = reference1
reference3 = reference1
```

Тепер екземпляр класу Person має три сильні посилання. Якщо знищити два з цих трьох посилань (включаючи і первісне посилання), присвоємо nil двом змінним, то залишиться одне сильне посилання, і екземпляр Person не буде звільнений:

```
reference1 = nil
reference2 = nil
```

ARC не звільнить екземпляр класу Person до тих пір, поки залишається останнє сильне посилання, знищивши яке вказуємо на те, що екземпляр більше не використовується:

```
reference3 = nil
// Prints "John Appleseed is being deinitialized"
```

Слабкі (weak) посилання

Слабкі посилання не утримуються за екземпляр, на який вони вказують, так що ARC не бере їх до уваги, коли підраховує посилання екземпляра. Такий підхід дозволяє уникнути ситуації, коли посилання стає

частиною циклу сильних посилань. Слабке посилання вказується ключовим словом weak перед ім'ям властивості або змінною, яка об’являється. Слабкі зв’язки використовуються циклів посилань, коли є ймовірність того, що в якийсь момент часу посилання матиме значення "nil".

Через те, що слабким посиланням дозволяється мати nil, що означає "відсутність значення", то потрібно оголошувати кожне слабке посилання як те, яка має опціональне значення. Опціональні типи – найбільш пріоритетний тип для відображення, тому що змінна може не мати значення в Swift.

Так як слабке посилання не сильно тримається за екземпляр, на який вказує, то можна звільнити екземпляр, на який вказує таке посилання. Таким чином ARC автоматично присвоює слабкому посиланню nil, коли екземпляр, на який воно вказує, звільняється (рис. 1) [1].

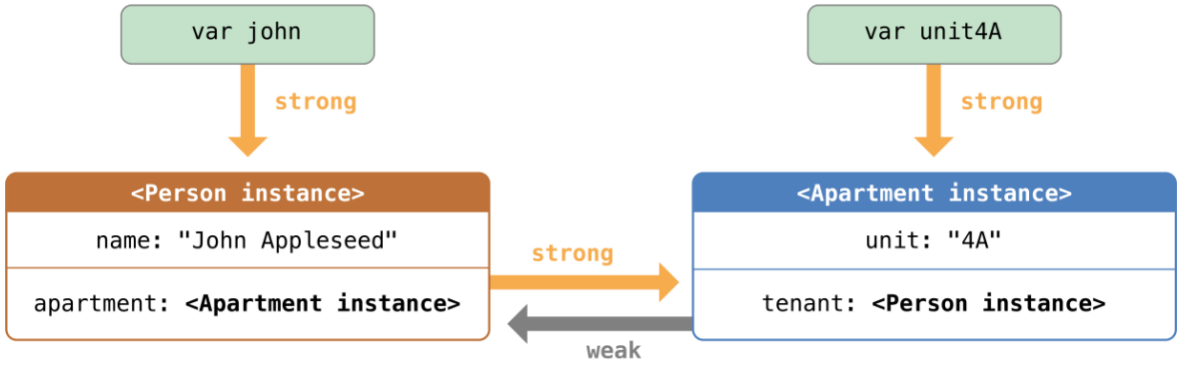

Рисунок 1 – Зв’язок між двома екземплярами

Невизначені (unowned) посилання

Як і слабкі посилання, невизначені посилання так само не мають сильного зв'язку з екземпляром, на який вони вказують. На відміну від слабких посилань, невизначені посилання завжди мають значення. Через це невизначені посилання мають неопціональний тип. Слабке посилання вказується ключовим словом unowned перед ім'ям властивості або змінною, яка об’являється.

Так як невизначене посилання не є опціональним, то не потрібно і розгортати (unwrap) його кожен раз, коли воно буде використовуватися. Тому до невизначеного посилання можна звертатися безпосередньо. Однак ARC не може встановити значення посилання на nil, коли екземпляр, на який воно посилається, звільнений, так як змінні неопціонального типу не можуть мати значення nil (рис. 2) [1].

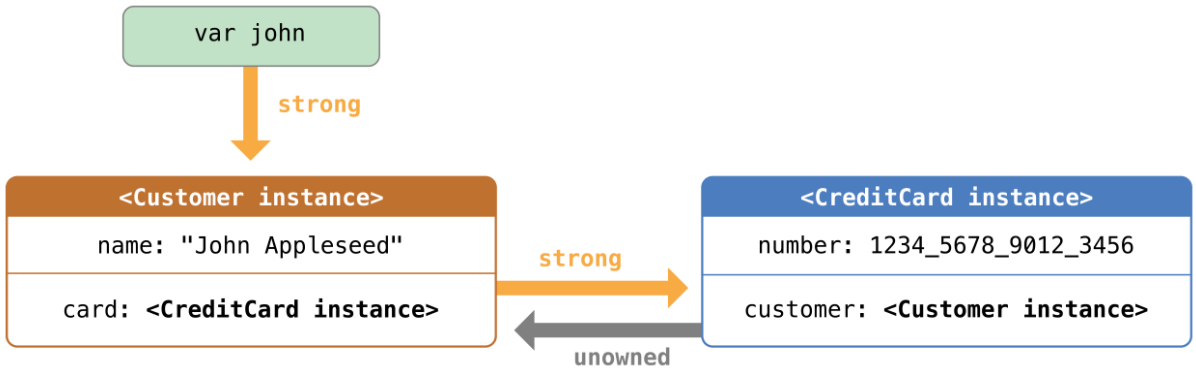

Рисунок 2 – Зв’язок між двома екземплярами

Висновок. Отже, Automatic Reference Counting на даний момент є потужним інструментом, який дозволяє зменшити витрати на управління пам’яттю і сфокусувати їх на основній логіці програмного продукту, що розробляється, адже саме автоматичне і високопродуктивне управління пам'яттю зможе підвищити продуктивність.

Література:

1. Офіційна документація мови програмування Swift. – Режим доступу: https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/. – Дата доступу 12.05.2016.
2. Офіційний сайт Apple для розробників. – Режим доступу: https://developer.apple.com/library/ios/releasenotes/ObjectiveC/RN-TransitioningToARC/Introduction/Introduction.html. – Дата доступу 21.06.2016.

3. Офіційний сайт блогу RayWenderlich. – Режим доступу: https://www.raywenderlich.com/2657/memory-management-tutorial-for-ios. – Дата доступу 04.05.2016.