23 Удаление экземпляров и ARC

Любой созданный экземпляр объектного типа данных, как и вообще любое хранилище вашей программы, занимает некоторый объем оперативной памяти. Если не производить своевременное уничтожение экземпляров и освобождение занимаемых ими ресурсов, то в программе может произойти утечка памяти.

ПРИМЕЧАНИЕ Утечка памяти — это программная ошибка, приводящая к излишнему расходованию оперативной памяти.

Одним из средств решения проблемы исключения утечек памяти в Swift является использование деинициализаторов, но возможности Swift не ограничиваются только этим.

23.1. Уничтожение экземпляров

Как мы отмечали в предыдущей главе, непосредственно перед уничтожением экземпляра класса вызывается деинициализатор, при этом остался нерассмотренным вопрос о том, как удаляется экземпляр. Любой экземпляр может быть удален одним из двух способов:

- его самостоятельно уничтожает разработчик;
- его уничтожает Swift.

Область видимости

Ранее мы самостоятельно уничтожали созданный экземпляр опционального типа `SubClass?`, передавая ему в качестве значения `nil`. Теперь обратимся к логике работы Swift. Для этого разработаем класс `myClass`, который содержит единственное свойство `description`. Данное свойство служит для того, чтобы отличать один экземпляр класса от другого.

Необходимо создать два экземпляра, один из которых должен иметь ограниченную область видимости (листинг 23.1).

Листинг 23.1

```
class myClass{
    var description: String
    init(description: String){
        print("Экземпляр \(description) создан")
        self.description = description
    }
    deinit{
        print("Экземпляр \(self.description) уничтожен")
    }
}
var myVar1 = myClass(description: "ОДИН")
if true {
    var myVar2 = myClass(description: "ДВА")
}
```

Консоль:

```
Экземпляр ОДИН создан
Экземпляр ДВА создан
Экземпляр ДВА уничтожен
```

Экземпляр `myVar2` имеет область видимости, ограниченную оператором `if`. Несмотря на то что мы не выполняли принудительное удаление экземпляра, для него был вызван деинициализатор, в результате он был автоматически удален.

Причина удаления второго экземпляра лежит в области видимости хранящей его переменной. Первый экземпляр инициализируется вне оператора `if`, а значит, является глобальным для всей программы. Второй экземпляр является локальным для условного оператора. Как только выполнение тела оператора завершается, область видимости объявленной в нем переменной заканчивается и она вместе с хранящимся в ней экземпляром автоматически уничтожается.

Количество ссылок на экземпляр

Рассмотрим пример, в котором на один экземпляр указывает несколько разных ссылок (листинг 23.2).

Листинг 23.2

```
class myClass{
    var description: String
    init(description: String){
        print("Экземпляр \(description) создан")
        self.description = description
    }
    deinit{
        print("Экземпляр \(self.description) уничтожен")
    }
```

```
10 }
11 var myVar1 = myClass(description: "ОДИН")
12 var myVar2 = myVar1
13 myVar1 = myClass(description: "ДВА")
14 myVar2 = myVar1
```

Консоль:

```
Экземпляр ОДИН создан
Экземпляр ДВА создан
Экземпляр ОДИН уничтожен
```

В переменной `myVar1` изначально хранится ссылка на экземпляр класса `myClass`. После записи данной ссылки в переменную `myVar2` на созданный экземпляр уже указывают две ссылки. В результате этот экземпляр удаляется лишь тогда, когда удаляется последняя ссылка на него.

Не забывайте, что экземпляры классов в Swift передаются не копированием, а по ссылке.

ПРИМЕЧАНИЕ Экземпляр существует до тех пор, пока на него указывает хотя бы одна ссылка.

23.2. Утечки памяти

Утечка памяти может привести к самым печальным результатам, одним из которых может быть отказ пользователей от вашей программы.

Пример утечки памяти

Рассмотрим ситуацию, при которой может возникнуть утечка памяти. Разработаем класс, который может описать человека и его родственные отношения с другими людьми. Для этого в качестве типа свойств класса будет указан сам тип (листинг 23.3).

Листинг 23.3

```
1  class Human {
2      let name: String
3      var child = [Human?]()
4      var father: Human?
5      var mother: Human?
6      init(name: String){
7          self.name = name
8      }
9      deinit {
10         print(self.name+"  — удален")
11     }
12 }
13 if 1==1 {
14     var Kirill = Human(name: "Кирилл")
```

```
15      var Olga = Human(name: "Ольга")
16      var Aleks = Human(name: "Алексей")
17      Kirill.father = Aleks
18      Kirill.mother = Olga
19      Aleks.child.append(Kirill)
20      Olga.child.append(Kirill)
21  }
```

На консоль ничего не выводится, хотя все операции выполняются в теле условного оператора, то есть в ограниченной области видимости. Нами создано три экземпляра, указаны перекрестные ссылки друг на друга, но эти экземпляры вовремя не удаляются.

Экземпляры остаются неудаленными, поскольку к моменту, когда их область видимости заканчивается, ссылки на объекты все еще существуют, и Swift не может понять, какие из ссылок можно удалить, а какие нельзя.

Это типичный пример утечки памяти в приложениях.

Сильные и слабые ссылки

Swift пытается не позволить программе создавать ситуации, приводящие к утечкам памяти. Представьте, что произойдет, если объекты, занимающие большую область памяти, не будут удаляться, занимая драгоценное свободное пространство. В конце концов приложение «упадет». Такие ситуации приведут к потере пользователей приложения.

Для решения описанной ситуации, когда Swift не знает, какие из ссылок можно удалять, а какие нет, существует специальный механизм, называемый *сильными* и *слабыми ссылками*. Все создаваемые ссылки на экземпляр по умолчанию являются сильными. И когда два объекта указывают друг на друга сильными ссылками, то Swift не может принять решение о том, какую из ссылок можно удалить первой. Для решения проблемы некоторые ссылки можно преобразовать в слабые.

Слабые ссылки определяются с помощью ключевых слов `weak` и `onowned`. Модификатор `weak` указывает на то, что хранящаяся в параметре ссылка может быть в автоматическом режиме заменена на `nil`. Поэтому модификатор `weak` доступен только для опционалов. Но помимо опционалов бывают типы данных, которые обязывают переменную хранить значение (все неопциональные типы данных). Для создания слабых ссылок на неопционалы служит модификатор `unowned`.

Перепишем пример из предыдущего листинга, преобразовав в слабые ссылки в свойствах `father` и `mother` (листинг 23.4).

Листинг 23.4

```
class Human {
    let name: String
    var child = [Human?]()
    weak var father: Human?
    weak var mother: Human?
    init(name: String){
        self.name = name
    }
    deinit {
        print(self.name+"  — удален")
    }
}
if 1==1 {
    var Kirill = Human(name: "Кирилл")
    var Olga = Human(name: "Ольга")
    var Aleks = Human(name: "Алексей")
    Kirill.father = Aleks
    Kirill.mother = Olga
    Aleks.child.append(Kirill)
    Olga.child.append(Kirill)
}
```

Консоль:

```
Алексей — удален
Ольга — удален
Кирилл — удален
```

В результате все три объекта будут удалены, так как после удаления слабых ссылок никаких перекрестных ссылок не остается.

Указанные ключевые слова можно использовать только для хранилища определенного экземпляра класса. Например, вы не можете указать слабую ссылку на массив экземпляров или на кортеж, состоящий из экземпляров класса.

23.3. Автоматический подсчет ссылок

Хотя в названии данной главы фигурирует аббревиатура ARC (Automatic Reference Counting — автоматический подсчет ссылок), но в ходе изучения мы еще ни разу к ней не обращались. На самом деле во всех наших действиях с экземплярами классов всегда участвовал механизм автоматического подсчета ссылок.

Понятие ARC

ARC в Swift автоматически управляет занимаемой памятью, удаляя неиспользуемые объекты. С помощью этого механизма вы можете

«просто заниматься программированием», не переключаясь на задачи, которые система решает за вас в автоматическом режиме.

Как уже неоднократно повторялось, при создании нового хранилища, в качестве значения которому передается экземпляр класса, данный экземпляр помещается в оперативную память, а в хранилище записывается ссылка на него. На один и тот же экземпляр может указывать произвольное количество ссылок, и ARC в любой момент времени знает, сколько таких ссылок хранится в переменных, константах и свойствах. Как только последняя ссылка на экземпляр будет удалена или ее область видимости завершится, ARC незамедлительно вызовет деинициализатор и уничтожит объект.

Таким образом, ARC делает работу со Swift еще более удобной.

Сильные ссылки в замыканиях

Ранее мы выяснили, что использование сильных ссылок может привести к утечкам памяти. Также мы узнали, что для решения возникших проблем могут помочь слабые ссылки.

Сильные ссылки могут также стать источником проблем при их передаче в качестве входных параметров в замыкания. Захватываемые замыканиями экземпляры классов передаются по сильной ссылке и не освобождаются, когда замыкание уже не используется. Рассмотрим пример. В листинге 23.5 создается пустое опциональное замыкание, которому в зоне ограниченной области видимости передается значение (тело замыкания).

Листинг 23.5

```
class Human{
    var name = "Человек"
    deinit{
        print("Объект удален")
    }
}
var closure : (() -> ())?
if true{
    var human = Human()
    closure = {
        print(human.name)
    }
    closure!()
}
print("Программа завершена")
```

Консоль:

```
Человек
Программа завершена
```

Так как условный оператор ограничивает область видимости переменной human, содержащей экземпляр класса Human, то, казалось бы, данный объект должен быть удален вместе с окончанием условного оператора. Однако по выводу на консоль видно, что экземпляр создается, но перед завершением программы его деинициализатор не вызывается.

Созданное опциональное замыкание использует сильную ссылку на созданный внутри условного оператора экземпляр класса. Так как замыкание является внешним по отношению к условному оператору, а ссылка сильной, Swift самостоятельно не может принять решение о возможности удаления ссылки и последующего уничтожения экземпляра.

Для решения проблемы в замыкании необходимо выполнить захват переменной, указав при этом, что в переменной содержится слабая ссылка (листинг 23.6).

Листинг 23.6

```
class Human{
    var name = "Человек"
    deinit{
        print("Объект удален")
    }
}
var closure : (() -> ())?
if true{
    var human = Human()
    // измененное замыкание
    closure = {
        [unowned human] in
        print(human.name)
    }
    closure!()
}
```

Консоль:

```
Человек
Объект удален
Программа завершена
```

Захватываемый параметр human является локальным для замыкания и условного оператора, поэтому Swift без проблем может самостоятельно принять решение об уничтожении хранящейся в нем ссылки.

В данном примере используется модификатор unowned, поскольку объектный тип не является опционалом.