РЕАЛИЗАЦИЯ ПАТТЕРНА SINGLETON НА ПЛАТФОРМЕ IOS ПРИ ИСПОЛЬЗОВАНИИ РУЧНОГО МЕТОДА УПРАВЛЕНИЯ ПАМЯТЬЮ

В.В.Николаенко И.Н.Цырельчук

Белорусский государственный университет информатики и радиоэлектроники, г.Минск, Республика Беларусь, uladzimir.nikalayenka@gmail.com

Abstract. In this article describes implementation of pattern singleton for iOS platform with use Objective-c programming language and manually memory management process. Described main features linked with app architecture implemented on Objective-C.

Паттерн Сиглтон – один из самых простых паттернов[1]. Основное предназначение этого паттерна в том, чтобы сделать объект экземпляра класса единственным в системе. В объектно-ориентированном приложении класс, реализованный паттерном Синглтон всегда возвращает один и тот же экземпляр самого себя. Тем самым обеспечивает единую точку доступа для ресурсов, которые представляет объект класса.

В iOS существует множество стандартных классов, которые реализованы по паттерну Синглтон: NSNotificationCenter, UIApplication, NSUserDefaults, UIDevice, NSFileManager и т.д.

В первую очередь для реализации паттерна необходимо объявить фабричный метод класса, который будет возвращать экземпляр. Так же создать static переменную, в которой будет хранится этот экземпляр (рис. 1).

```
@implementation Singleton

static Singleton *sharedSingleton_ = nil;

+ (Singleton *) sharedInstance {
  if (sharedSingleton_ == nil)
  {
    sharedSingleton_ = [[Singleton alloc] init];
  }
  return sharedSingleton_;
}
@end
```

Рисунок 1 – Имплементация фабричного метода

Реализация этого паттерна на языке Objective-C имеет некоторые особенности, связанные с этим языком. Любой метод в Objective-C является открытым, а сам язык является динамически типизированным, поэтому любой объект может послать абсолютно любое сообщение другому объекту (вызвать метод)[2]. Поэтому необходимо решить две проблемы, чтобы можно было использовать это синглтон в реальном коде:

- вызывающий объект не может создать объект синглтона через другие средства выделения памяти
- ограничения на создание объекта должны быть согласованы с моделью подсчета ссылок.

Для этого класс синглтона должен реализовать несколько методов для управления памятью (рис. 2). В методе allocWithZone:(NSZone*) происходит просто возврат экземпляра класса. Фактически метод alloc вызывает для выделения памяти allocWithZone:(NSZone *) с зоной NULL.

```
@implementation Singleton

static Singleton * sharedSingleton_ = nil;

+ (Singleton*) sharedInstance
{
  if (sharedSingleton_ == nil)
  {
    sharedSingleton_ = [[super allocWithZone:NULL] init];
  }
  return sharedSingleton_;
}

+ (id) allocWithZone:(NSZone *)zone {
  return [[self sharedInstance] retain];
}

- (id) copyWithZone:(NSZone*)zone {
  return self;
}

- (id) retain {
  return self;
}
- (NSUInteger) retainCount {
  return NSUIntegerMax;
}
- (void) release {
  // do nothing
}
- (id) autorelease {
  return self;
}
@end
```

Рисунок 2 – Имплементация фабричного метода исправленная

Аналогично нужно переопределить метод copyWithZone:(NSZone*)zone, чтобы убедиться, что он не вернет копию экземпляра, а вернет тот же самый, возвращая self.

Другие методы, такие, как retain, release и autorelease, переопределяются, чтобы убедиться, что они не сделают ничего (в модели управления памятью с подсчетом ссылок), кроме как вернут self.

Метод retainCount возвращает NSUIntegerMax(4 294 967 295) для предотвращения удаления экземпляра из памяти в течение жизни приложения.

Описанная в примере реализация паттерна синглтон подходит лишь для общего пользования. Для использования ее в многопоточном приложении необходимо добавить экземпляры NSLock или блоки @synchronized() вокруг проверки статической перемнной на nil.

Так же можно использовать Grand Central Dispatch (GCD) как показано в примере на рисунке 3.

```
+(Singleton *) sharedInstance
{
  static Singleton * sharedInstance= nil;
  static dispatch_once_t once_token = 0;
  dispatch_once(&once_token, ^
                {
                  sharedInstance =  [Singleton  new];
                });
  return sharedInstance ;
}
```

Рисунок 3 – Имплементация фабричного метода с использование GCD

Литература

1. Chung.C. Pro Objective-c Design Patterns / C.Chung – NY, Apress 2012.
2. Galloway M. Effective Objective-C 2.0 / M.Galloway – NJ, Pearson Education 2013.