ISSN No. 0976-5697

Volume 5, No. 3, March-April 2014

International Journal of Advanced Research in Computer Science

REVIEW ARTICLE

Available Online at www.ijarcs.info

Singleton Design Pattern in iOS Application Development

Md. Shakural Islam Sarder
Department of Computer Science & Engineering,
Shahjalal University of Science and Technology,Sylhet
Dhaka, Bangladesh

Nikson Kanti Paul
Department of Computer Science & Engineering,
Shahjalal University of Science and Technology, Sylhet
Dhaka, Bangladesh

Abstract: Software design patterns help structure code, solve problems and affect the design of programs by recognizing the possible abstractions in the problem. Design patterns are important technique for improving software quality since they address a fundamental challenge in large scale software development. In a rapidly evolving system lack of using design pattern efficiently or it's scarcity is a huge problem when design decisions affect the existing architecture by developing new feature to the existing system, modifying existing system or maintaining existing system[1]. Singleton design pattern is the most widely used design pattern in iOS application development.This paper describes the important role of Singleton Design Pattern in iOS application development and gives us some insight into how Singleton Design Pattern are implemented in some of the modern middleware.

Keywords: Design pattern ,Singleton design pattern, iOS, Mobile application.

I. INTRODUCTION

Software Design Patterns are reusable solutions to common problems in software design. Building a system from scratch requires a carefully made design and this is where design patterns come handy. Various design patterns are available to support development process. Each pattern provides the main solution for particular problem.

Generally developers use their own design for a given software requirement, which may be new each time they design for a similar requirement. Such method is time consuming and makes software maintenance more difficult. Adapting the software application to future requirements changes will be difficult if adequate care is not taken during design. Adopting Design Pattern while designing applications can promote reusability and consistency. Without the adoption of the design patterns or wrong selection of them will make the development of the application more complex and hard to be maintained. Thus knowledge in design patterns is essential in selecting the most appropriate patterns in the iOS application development. Unfortunately, the ability of the developers in applying the design pattern is undetermined. The good practice of using design patterns is encouraged and efforts have to be taken to enhance the knowledge on design patterns[2].

There are some cases in which it makes sense to have exactly one instance of a class. The Singleton design pattern ensures that only one instance exists for a given class and that there's a global access point to that instance. Singleton class uses lazy loading to create the single instance.[7]

II. SINGLETON DESIGN PATTERN

In iOS application development, sharing data between views are frequently required and it can be easily done by using singleton class. Singleton class provide global access. The Singleton model instantiate just one instance of a class which doesn't allow a second instance of that same class to be loaded anywhere in the application. If we try to create another instance then it will get referred to the already created instance. This prevents data from being mashed up.

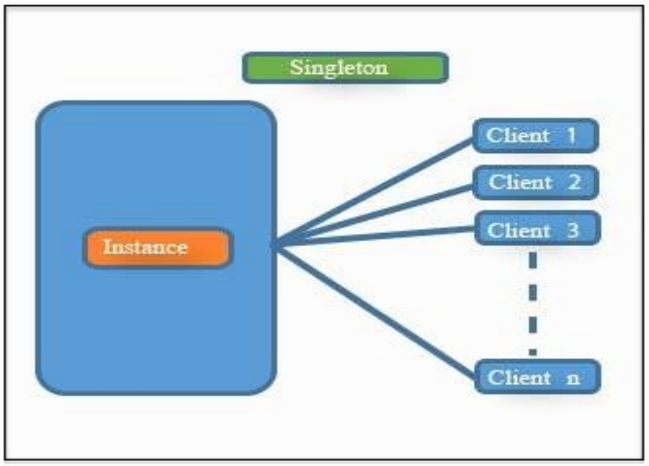

Figure 1: Singleton Design Pattern

The pattern provides the flexibility such that any of its subclasses can override the instance method and have total control over object creation of itself without changing code in the client. Or even better, the instance implementation in the parent class can handle dynamic object creation. The actual type of a class can be determined to make sure the correct object is created at runtime. Many of the core Apple classes follow this pattern. Some of the common approach that Apple uses are given below: [8][9][11]
[NSUserDefaults standardUserDefaults],
[UIApplication sharedApplication],
[UIScreen mainScreen],
[NSFileManager defaultManager]
all return a Singleton object.[3]

A. When to use Singleton design pattern:

The Singleton pattern provides a well-known access point to client classes that want to create a unique instance of and access to a shared resource. Although a static global object reference or a class method can provide a global access point, the global object cannot prevent the class

getting instantiated more than once, and the class method lacks the flexibility of decoupling. There are some scenario in which singleton design pattern is much more preferable. Consider the following cases

a. There must be exactly one instance of a class that must be accessible from all well-known access point.
b. The sole instance of the singleton class can be extended only by subclassing, and it won't break client code with the extended object.

Different managers used to load resources (e.g. texture managers, sound managers), user input managers, and also objects which need to keep track of global game information (e.g. game score) are good candidates for singleton classes. Because of its purpose, singletons are often given a global scope, and this is why the Singleton can be perceived both as pattern and anti-pattern[3].

B. Implementation of Singleton class:

The canonical way of declaring Singleton Class in objective C as follows [8][9][11]

MySingletonClass.h

```
#import <Foundation/Foundation.h>

@interface MySingletonClass : NSObject{
    NSString *userName;
}

@property(nonatomic,retain) NSString *userName;

+ (MySingletonClass *) sharedInstance;

@end
```

Figure 2: Declaration of singleton class

MySingletonClass.m

```
#import "MySingletonClass.h"

static MySingletonClass *mySharedInstance = nil;

@implementation MySingletonClass

@synthesize userName;

+ (MySingletonClass *) sharedInstance{
    @synchronized(self) {
        if(mySharedInstance == nil)
            mySharedInstance = [[super allocWithZone:NULL] init];
    }
    return mySharedInstance;
}

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

- (id)init {
    if (self = [super init]) {
        userName = @"Barak Obama";
    }
    return self;
}

- (void)dealloc {

}

@end
```

Figure 3: Implementation of Singleton Class(In ARC enabled environment)

MySingletonClass.m

```
#import "MySingletonClass.h"

static MySingletonClass *mySharedInstance = nil;

@implementation MySingletonClass

@synthesize userName;

+ (MySingletonClass *) sharedInstance{
    @synchronized(self) {
        if(mySharedInstance == nil)
            mySharedInstance = [[super allocWithZone:NULL] init];
    }
    return mySharedInstance;
}
-(void) myCustomMethod{
    NSLog(@"Custom Method Called");
}
- (id)copyWithZone:(NSZone *)zone {
    return self;
}
- (id)init {
    if (self = [super init]) {
        userName = @"Barak Obama";
    }
    return self;
}
+ (id)allocWithZone:(NSZone *)zone {
    return [[self sharedInstance] retain];
}
- (id)retain {
    return self;
}
- (unsigned)retainCount {
    return UINT_MAX;
}
- (id)autorelease {
    return self;
}
- (void)dealloc {
    [userName release];
    [super dealloc];
}

@end
```

Figure 4: Implementation of Singleton Class(In ARC disabled environment)

In Figure 2, A singleton class name ***MySingletonClass*** is declared. A string name ***userName*** has added which is the member variable of ***MySingletonClass.*** Later we will show how we will access this member variable.

In figure 3, shows the implementation of singleton class ***mySingletonClass in ARC enabled environment.*** According to apple documentation, Automatic Reference Counting (ARC) is a compiler feature that provides automatic memory management of Objective-C objects. Rather than having to think about retain and release operations, ARC allows you to concentrate on the interesting code, the object graphs, and the relationships between objects in your application[9].

In Figure 4, shows the implementation of singleton class ***mySingletonClass in ARC disabled environment.***

There are a few methods needs to clarify in order to better understanding of singleton class. [4][5][6]

(a). **+ (MySingletonClass *) sharedInstance** get the shared instance and create it if necessary.
(b). **- (void) myCustomMethod** is sample method. User will create their own method based on requirement.
(c). **- (id)copyWithZone:(NSZone *)zone** prevent to generate multiple copies of the singleton.
(d). **- (id) init** is init method, that will get called the first time the Singleton is used.
(e). **- (id)allocWithZone:(NSZone *)zone** prevent to allocate a new instance, so return the current instance.
(f). **- (id)retain**, don't require a retain counter for this object. Just return the object.

(g). - **(unsigned)retainCount** make sure that this object will never release.

(h). - **(unsigned)autorelease,** don't require any action. Just return the object.

(i). - **(void)dealloc** will never be called, as the singleton survives for the duration of the application.

Finally, Singleton class has been implemented and ready to use for development. But singleton's initialization method could be called more than once. Its possible that singleton class can be called concurrently from different threads. In order to overcome this problem, Grand Central Dispatch (Cocoa and iOS4+) approach is used to ensure singleton is never executed more than once even when multiple threads access sharedInstance concurrently[3].Figure 5, shows the thread safe implementation of singleton class.

```
10
11 + (MySingletonClass *) sharedInstance{
12     static dispatch_once_t onceToken;
13     dispatch_once(&onceToken, ^{
14         mySharedInstance = [[self alloc] init];
15     });
16     return mySharedInstance;
17
18 }
19
```

Figure 5 : Thread safe implementation of singleton class

III. ACCESSING SINGLETON CLASS

Singleton design pattern provide global access.So the instance of the singleton class that exist in the application life cycle can be accessed from anywhere of the code base.

We can access singleton class by using following fashion:

MySingletonClass *singleton = [MySingletonClass sharedInstance] [singleton myCustomMethod]

Or

[[MySingletonClass sharedInstance] myCustomMethod] [6]

Here myCustomMethod is the method that is implemented in the singleton class file.

III. FUTURE WORK

A singleton class always returns the same instance of itself that provides a global access point for the resources provided by the object of the class which is an extremely powerful way of sharing data between different classes. However there are some scenarios in which singleton design pattern should be avoided. Singletons are global and create direct implementation dependencies and can get complicated when it comes to unit testing. Also it holds the memory of the application life cycle. Future work is to determine the cases in which singleton design pattern should be used or should be avoided and design a hybrid design model that will resolve those issues.

IV. CONCLUSION

It is evident that singleton design pattern plays an important role in developing iOS application.In particular, Singleton is widely used by Apple. Although the main purpose of the Singleton is to limit number of instances to one single instance, it is mainly used to allow a global access to the class variables. This paper first presented an overview of the singleton design pattern and it's implementation procedure. The paper then went on to elaborate on some of the common scenario in which singleton design pattern can be used. Despite the controversial usage of singleton design pattern, the concept of globally accessed objects comes in handy. All though there are multiple alternatives to Singleton design pattern technique, the appropriateness of the Singleton depends on the design decision that must be made and the consequences of that decision must be followed and understood.

V. REFERENCES

[1] P´eter Hegedu □s, D´enes Ba´n, Rudolf Ferenc, and Tibor Gyim´othy, "Myth or Reality? Analyzing the Effect of Design Patterns on Software Maintainability", University of Szeged, Department of Software Engineering A´rpa´d t´er 2. H-6720 Szeged, Hungary.

[2] Vijay k kerji, "ABSTRACT FACTORY AND SINGLETON DESIGN PATTERNS TO CREATE DECORATOR PATTERN OBJECTS IN WEB APPLICATION", International Journal of Advanced Information Technology (IJAIT) Vol. 1, No.5, October 2011.

[3] Astahovs Ilja, "Use of design patterns for mobile game development", UMEA University.

[4] Brian d foy, "The Singleton design pattern," The Perl Review (0, 1 b 112) · 36.

[5] Mat Galloway, Singletons in Objective-C.

[6] Saurabh Sharma, "Singleton Design Pattern for Objective - C", Jun 12, 2011.

[7] Ray Wenderlich, Tutorials for developers and gamers, "iOS design pattern", september 4, 2013.

[8] iOS developer library, "Singleton".

[9] iOS developer library, "Transitioning to ARC Release Notes".

[10] Insanely Crazy Development, "Using the Singleton Design Pattern in Objective-C",January 06, 2012.

[11] iOS developer library, "App design basics".