# Basic Exercises Part 5.3. Design Patterns. Singletons

## Singletons Introduction

* A singleton is a special kind of class where only one instance of the class exists for the current process. (In the case of an iPhone app, the one instance is shared across the entire app.). A singleton class returns the same instance no matter how many times an application requests it. Unlike a regular class, A singleton object provides a global point of access to the resources of its class.
* Some examples in UIKit are:
  + [UIApplication sharedApplication] Which returns the sole instance of the application itself.
  + [NSFileManager defaultManager] Which returns the file manager instance. Singletons can be an easy way to share data and common methods across the entire app.
* Rather than create instances of the singleton class using alloc/init, you'll call a class method that will return the singleton object. You can name the class method anything, but common practice is to call it shared*Name* or default*Name*.
* When to use a Singleton?

Singletons are used in situations where this single point of control is desirable, such as with classes that offer some general service or resource.

### **1.1 How to create Singleton Classes**

First, create a New file and subclass it from NSObject. Name it anything, we will use CommonClass here. Xcode will now generate CommonClass.h and CommonClass.m files for you.

In your CommonClass.h :

#import <Foundation/Foundation.h>

**@interface** CommonClass : NSObject {

}

+ (CommonClass \*)sharedObject;

**@property** NSString \*commonString;

**@end**

In your CommonClass.m :

#import "CommonClass.h"

**@implementation** CommonClass

+ (CommonClass \*)sharedObject {

**static** CommonClass \*sharedClass = **nil**;

**static** dispatch\_once\_t onceToken;

dispatch\_once(&onceToken, ^{

sharedClass = [[**self** alloc] init];

});

**return** sharedClass;

}

- (**id**)init {

**if** (**self** = [**super** init]) {

**self**.commonString = @"this is string";

}

**return** **self**;

}

**@end**

### **1.2 How to use singleton classes**

The Singleton Class that we created earlier will be accessible from anywhere in the project as long as you have imported CommonClass.h file in the relevant module. To modify and access the shared data in Singleton Class, you will have to access the shared Object of that class which can be accessed by using sharedObject method like following:

[CommonClass sharedObject]

To read or modify the elements in Shared Class, do the following:

NSString \*commonString = [[CommonClass sharedObject].commonString; //Read the string in singleton class

NSString \*newString = @"New String";

[CommonClass sharedObject].commonString = newString;//Modified the string i

### **1.3 Challenge**

Create a simple app. Implement two view controllers. Pass data back from controller B (second) to controller A (main). Use the UI elements you prefer. Implement the solution using the Singleton pattern. Use Objective C.

# Summary

Singleton classes are an important concept to understand because they exhibit an extremely useful design pattern. This idea is used throughout the iPhone SDK, for example, UIApplication has a method called sharedApplication which when called from anywhere will return the UIApplication instance which relates to the currently running application.