Objective-C

Курс лекций и семинаров для студентов,
желающих научиться программировать под iPhone

Осень-Зима 2013

Лекция №7

Patterns: Model-view-controller and friends

Автор: Дмитрий Волков, iPhone Developer, Sibers

Sibers®

Что вы узнаете сегодня?

- Model-View-Controller
- Delegation
- Singleton
- Observer
- Class Cluster

Patterns: Thousands of them

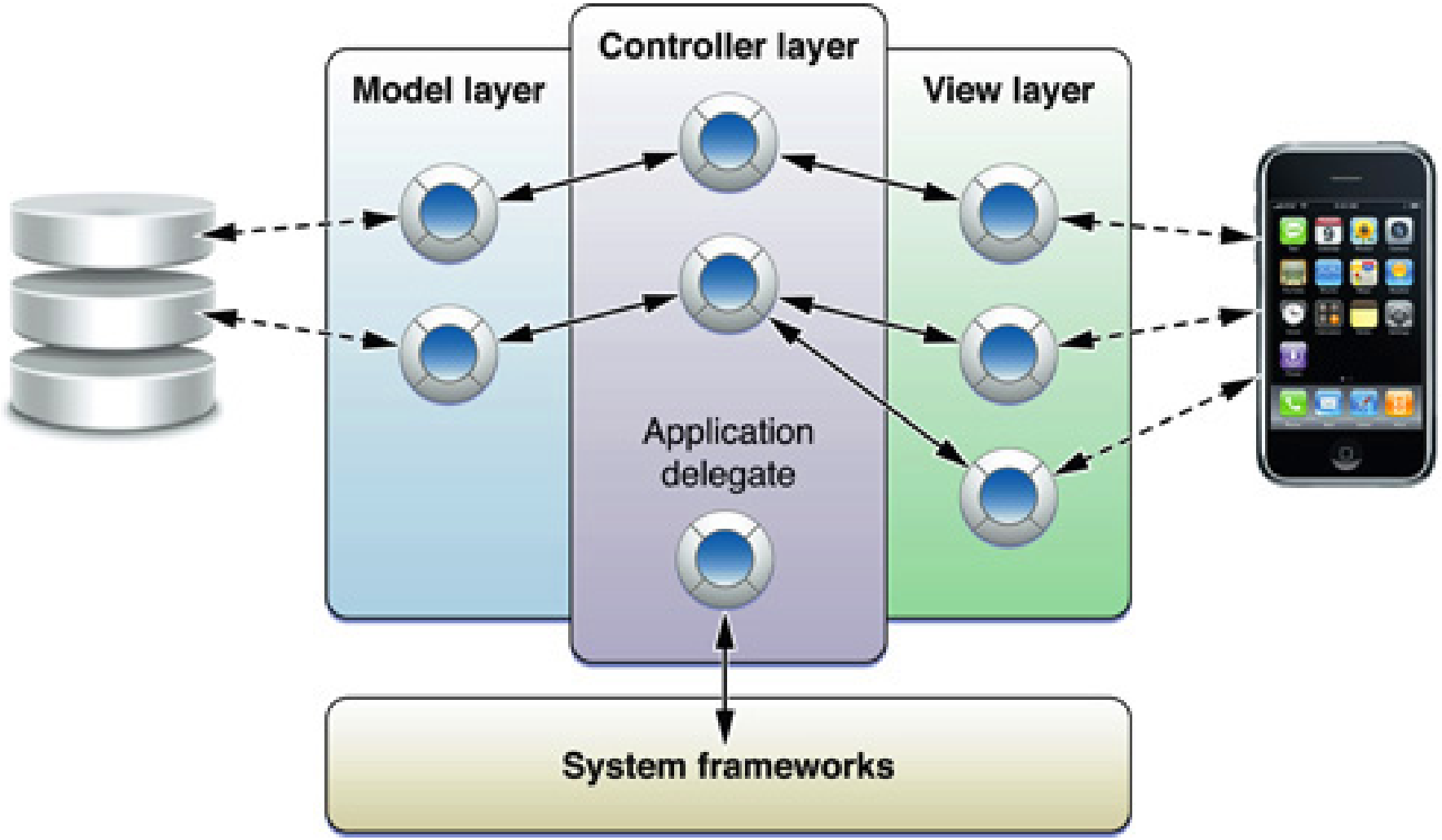
- «Gang of Four» — «**Design Patterns**: Elements of Reusable Object-Oriented Software».
- ***Model-View-Controller***
  - Model-View-Presenter
  - Model-View-ViewModel
- ***Database patterns***
  - Active Record
  - Data Access Object
- ***Concurrent patterns***

Patterns: Cocoa-wide

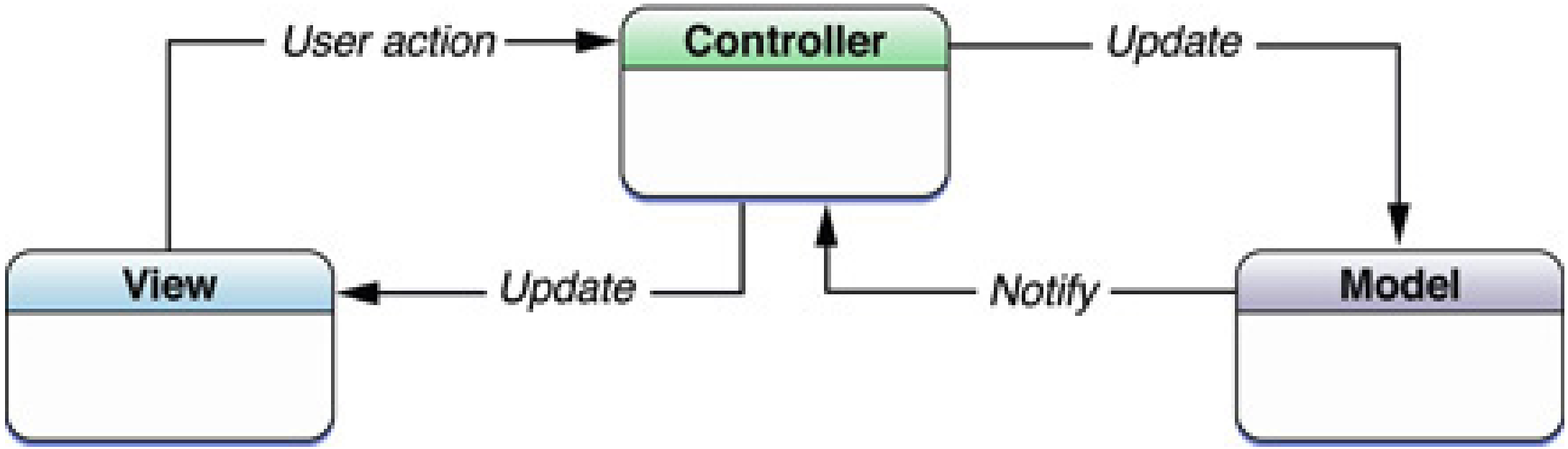
Meta : Model-View-Controller

- Singleton
- Observer
- Delegation
- Action-target
- Responder chain
- Factory
- Class Clusters

Model-View-Controller



Model-View-Controller



Model-View-Controller: Model Layer

Модель — данные специфичные для приложения а так же методы их обработки, валидации и изменения.

- Модель **НЕ** зависит и **НЕ** связана с View.
- По возможности, модель должна быть максимально переносимой.
- Модель как-либо сообщает о своих изменениях контроллеру, который обновляет состояние View.
- Большую часть модельных объектов предоставляет фреймворк Foundation.

Model-View-Controller: View layer

View (aka Представление) — часть приложения, с которым работает пользователь.

- Отображают данные модели в виде, понятном и доступном для изменения пользователю.
- Обрабатывают взаимодействие пользователя с приложением.
- View **НЕ** связаны и **НЕ** зависят (ну, почти) от Модели.
- View сообщают контроллерам о том, что именно изменил пользователь в UI.

Model-View-Controller: Controller layer

Контроллеры — прослойка между моделью и представлением данных пользователю.

Контроллеры — основной связующий слой:

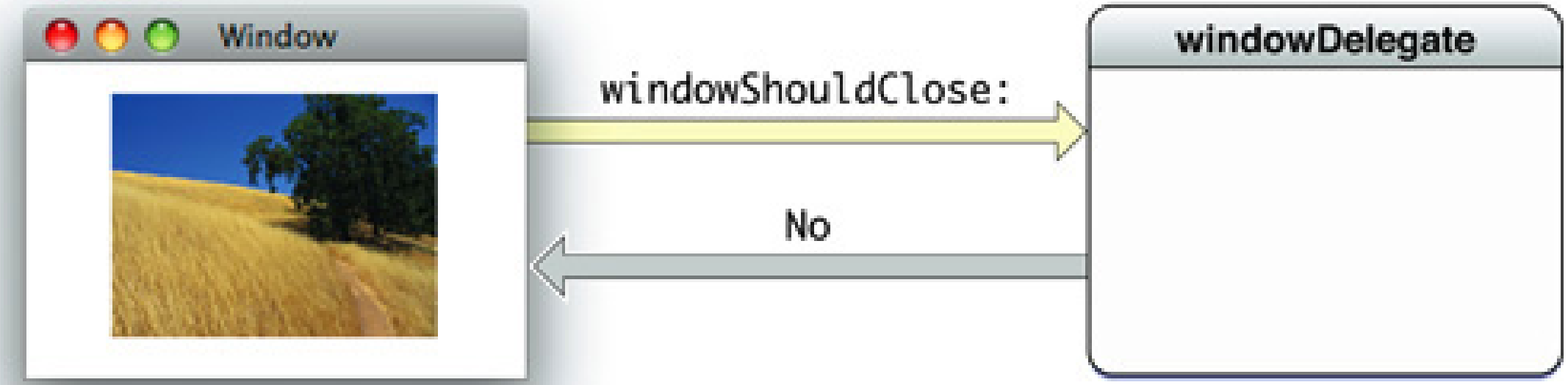
- Получают данные из модели, и передают их представлению.
- Оповещаются представлением об изменении данных пользователем и обновляют состояние модели.

Контроллеры так же могут следить за обновлениями данных в модели, если она как либо обновляет свое внутреннее состояние, и передает эти изменения представлениям.

Delegation

- Одним из основных коммуникационных паттернов в Cocoa / Cocoa Touch является delegation.
- Класс делегирует часть своей функциональности какому-либо другому объекту, и запрашивает у него необходимые данные либо посылает сообщения об изменении своего состояния.
- Делегирование позволяет разграничивать обязанности классов и поддерживать слабую связанность между объектами.
- В Objective-C делегирование реализуется посредством протоколов.

Delegation



Delegation

Обычно, делегирование в Objective-C реализуется с помощью протоколов.

Так как протоколы могут описывать обязательные и не обязательные для реализации методы (@required / @optional), в Cocoa выделяют 2 типа делегирования:

- ***Delegate*** — объект оповещает своего делегата о каких-либо событиях. (обычно - ***@optional*** методы)
- ***Datasource*** — объект запрашивает у своего делегата данные, необходимые для работы. (обычно - ***@required*** методы)

Delegation: Implementation

```
@protocol UITableViewDataSource<NSObject>

@required

- (NSInteger)tableView:(UITableView *)table numberOfRowsInSection:(NSInteger)section;

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath;


@optional

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView;

...

@end


@interface UITableView : UIScrollView

....

@property(nonatomic, assign) id<UITableViewDataSource> datasource.

...

@end
```

```
@implementation UITableView
...

- (void) layoutCells
{
	NSInteger sectionsCount = 1;
	if([self.dataSource respondsToSelector:@selector(numberOfSectionsInTableView:)])
	{
		sectionsCount = [self.dataSource numberOfSectionsInTableView:self];
	}
	for(int i = 0; I < sectionsCount; i++)
	{
		int rowsCount = [self.dataSource numberOfRowsInSection:i];
		...
	}

}
@end
```

```
@interface TableViewDataSource:NSObject<UITableViewDatasource>
@end

@implementation TableViewDataSource
- (NSInteger)tableView:(UITableView *)table numberOfRowsInSection:(NSInteger)section
{
        return 10;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath
*)indexPath
{
        return [UITableViewCell new];
}
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
        return 5;
}

@end
```

Delegation: Meanwhile in ViewController

```
- (void) viewDidLoad
{
    UITableView* tableView = [UITableView new];
    tableView.frame = CGRectMake(0, 0, 100, 100);
    [self.view addSubView:tableView];
    _tableDataSource = [TableDataSource new];
    tableView.dataSource = _tableDataSource;
}
```

Delegation: Retain cycles

- ВАЖНО: при реализации делегирования важно помнить, что @property, в которые вы присваиваете объект-делегат, ВСЕГДА должны иметь модификатор assign (weak).
- Системные классы так же придерживаются этого правила (за парой исключений)
- Так как, обычно, делегат хранит сильную ссылку (***strong reference***) на объект-делегатор:
  - Если объект-делегат и объект-делегатор будут хранить сильные ссылки друг на друга, то оба объекта ***НИКОГДА*** не будут удалены из памяти — retain cycle.
  - Для предотвращения таких проблем используются слабые ссылки (***weak reference***).

Singleton: One to rule them all

Singleton — паттерн, гарантирующий, что в приложении существует только один экземпляр объекта заданного класса.

- Так же довольно широко используется в Cocoa / Cocoa Touch.
- Предоставляет глобальную точку доступа к объекту.
- По своей сути глобальная переменная.
- Позволяет синхронизировать доступ к каким-либо ограниченным ресурсам (камера, микрофон, акселерометр и т.д.)
- Довольно часто модель в iOS реализуется посредством паттерна Singleton.

Singleton: One to rule them all

- Singleton ограничивает возможность создания объектов данного класса.
- Переопределяются методы alloc / init для того, чтобы возвращать указатель на один и тот же объект.
- Так же, переопределяются методы управления памятью для невозможности удаления объекта.

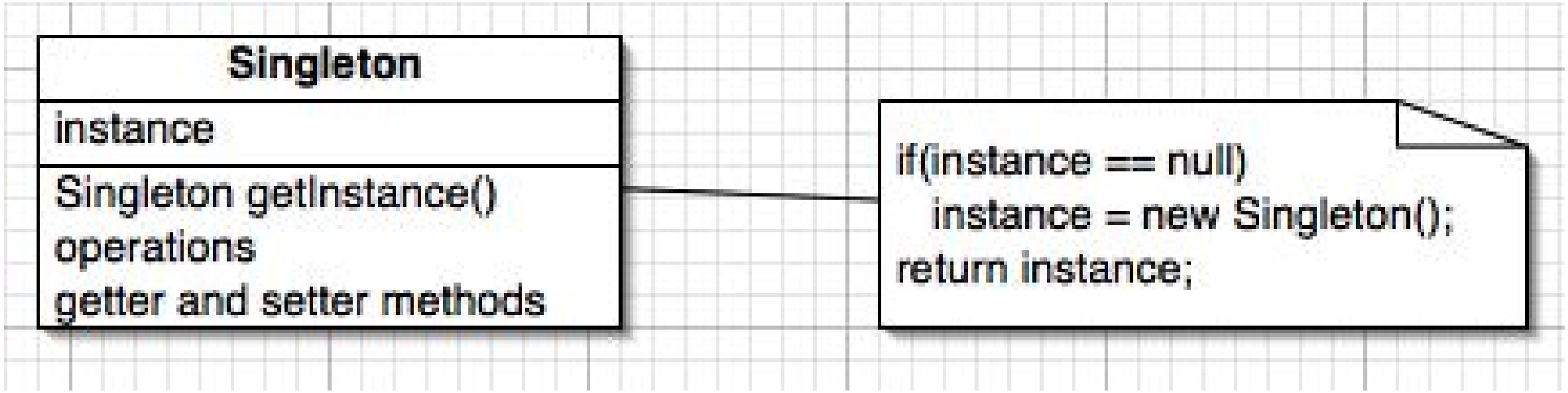
- В Objective C принято создавать метод класса в формате

  + (id) sharedInstance;

  + (id) shared<ClassName>;

- Singleton существует на протяжении жизни приложения, он не может быть удален.

Singleton: One to rule them all



```
//ARC swaggy implementation
@interface Model : NSObject
@property(nonatomic, copy) NSString* username;
+ (id) sharedModel;
...
@end

@implementation Model
+ (id) sharedModel
{
        static id instance = nil;
        static dispatch_once_t onceToken;
         dispatch_once(&onceToken, ^{
                instance = [[self alloc] init];
        });
        return instance;
}
@end
```

Singleton: Implementation notes

Приведенный выше пример кода — минимальная реализация паттерна Singleton, без переопределения методов инициализации и управления памятью.

- Реализация верна и для ARC, и для не-ARC кода.
- Для синхронизации при создании объекта используется функция dispatch_async из библиотеки GCD.

- Подразумевается, что вы, как разработчик приложения, будете получать указатель на синглтон посредством метода sharedModel.
- Полная Old-school реализация синглтона — http://habrahabr.ru/post/198470/

Singleton: Meanwhile in ViewController

```
- (void) viewDidLoad
{
    Model* model = [Model sharedModel];
    model.username = @”User”;
}

//somewhere else in the app
- (void) printUsername
{
    Model* model = [Model sharedModel];
    NSLog(@”%@”,model.username);//prints “User”
}
```

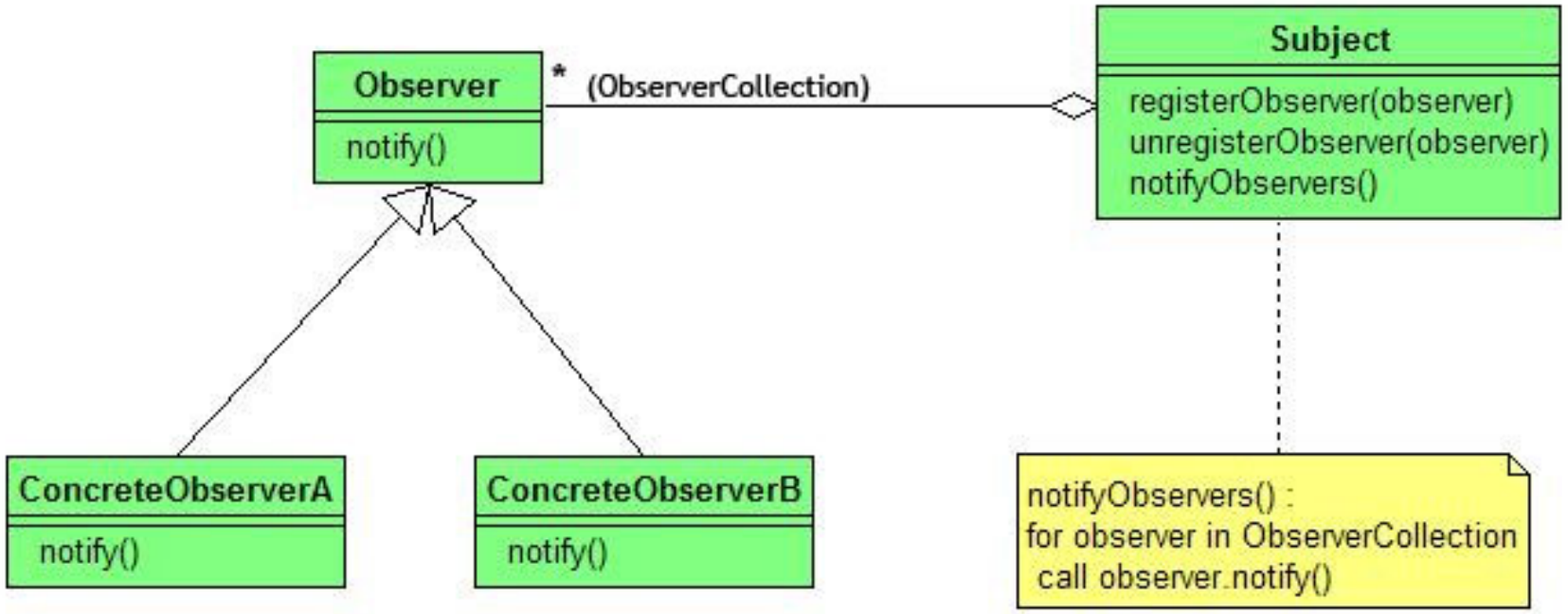
Observer: Basics

Observer — коммуникационный паттерн, позволяющий объектам подписываться наблюдателями изменения состояния другого объекта. При изменении состояния, наблюдаемый объект оповещает объекты, подписавшиеся на наблюдение за ним.

Делегирование — частный случай паттерна Observer, где только один объект (делегат) извещается об изменениях состояния наблюдаемого объекта.

В Сосоа существует встроенный класс, реализующий данный паттерн — ***NSNotificationCenter***

Observer: Humdrum diagram



Observer: Cocoa

- В Cocoa существует готовая реализация паттерна Observer — ***NSNotificationCenter***.

- ***NSNotificationCenter*** — объект-синглтон, позволяющий объектам подписываться на какие-либо события, используя строки как идентификаторы.

- Системные фреймворки рассылают множество нотификаций об изменении состояния системы / приложения, на которые вы можете подписаться и обработать нужным вам образом.

Observer: NSNotificationCenter.

- Основным объектом является класс NSNotificationCenter — используя его, вы можете подписываться на оповещение о событиях, отписываться от них, а так же рассылать их.

- ***NSNotification*** — объект, описывающий оповещение. Содержит имя оповещения, объект, который создал сообщение, а так же NSDictionary с дополнительной информацией.

- ***- (void)addObserver:(id)notificationObserver selector:(SEL)notificationSelector name:(NSString *)notificationName object:(id)notificationSender***

```
- (void) viewWillAppear:(BOOL) animated
{
    [[NSNotificationCenter defaultCenter] addObserver: self
selector:@selector(keyboardWasShown:) name:UIKeyboardDidShowNotification
object:nil];
}

- (void) viewWillDisappear:(BOOL) animated
{
    [[NSNotificationCenter defaultCenter] removeObserver:self];
}

- (void) keyboardWasShown:(NSNotification*) notification
{
    //...
}
```

Class Cluster: Abstract Factory

Class cluster — паттерн, используемый в Cocoa для порождения конкретных приватных подклассов абстрактного класса-родителя.

- Использует паттерн Abstract Factory.
- Для каждой отдельной задачи в Cocoa существует приватный, недоступный для разработчиков класс, создаваемый системой для более эффективной реализации конкретных функций.
- Облегчает работу с объектами, описывая единый для них интерфейс, но затрудняет расширяемость, так как вы наследуете абстрактный класс.

Class Cluster: Abstract Factory

Стандартные class-clusters:

- NSNumber
- NSArray
- NSDictionary
- NSData
- NSString
- И многие другие.

При наследовании и создании собственных классов-наследников, вам предстоит переопределить «примитивные» методы класса-родителя.

Class Cluster: Cocoa class-clusters

```
NSString* constString = @"String";//NSCFConstrantString
NSString* pathString = NSHomeDirectory();//NSPathStorage
NSString* string = [NSString stringWithFormat:@"%d",10]; //NSCFString

NSNumber* number = [NSNumber numberWithInt:10]; //NSCFNumber
NSNumber* bool = [NSNumber numberWithBool:YES]; //NSCFBoolean

UIButton* button = [UIButton
buttonWithType:UIButtonTypeRoundedRect];//UIRectButton
UIButton* customButton = [UIButton
buttonWithType:UIButtonTypeCustom];//UIButton
```

Заключение

MVC — основной паттерн, использующийся во фреймворках Cocoa.

MVC — концепция, описывающая, как вам следует разделять ваши данные и взаимодействие между ними.

Каждый из фреймворков Cocoa представляет одну из частей MVC — Foundation, содержит модельные объекты, AppKit / UIKit — представления и контроллеры.

Также, в системе широко используются паттерны Singleton, Delegation, Observer, Abstract Factory.

Спасибо за внимание!

Жду ваших вопросов.