

Developing iOS apps using objective-c

Presented By
Abdelrahman Sayed



Lecture 4



Agenda

- Blocks.
- Animated Effects Using UIKit Dynamics.
- CocoaPods.
- Using third party library (SDWebImage)



Blocks



Why Blocks ?

- A great and growing number of Apple's framework classes take blocks as parameters.
- Blocks are used as completion, error, and notification handlers, for sorting and enumeration, and for view animations and transitions.
- Simplify and reduce the amount of code you have to write.
- Your code is much more readable, less cluttered, more concise, and easier to understand.



Syntax

```
int (^myMultiplier) (int, int) = ^int (int a, int b){  
    NSLog(@"From inside block");  
    return a * b;  
};
```



Syntax Cont.

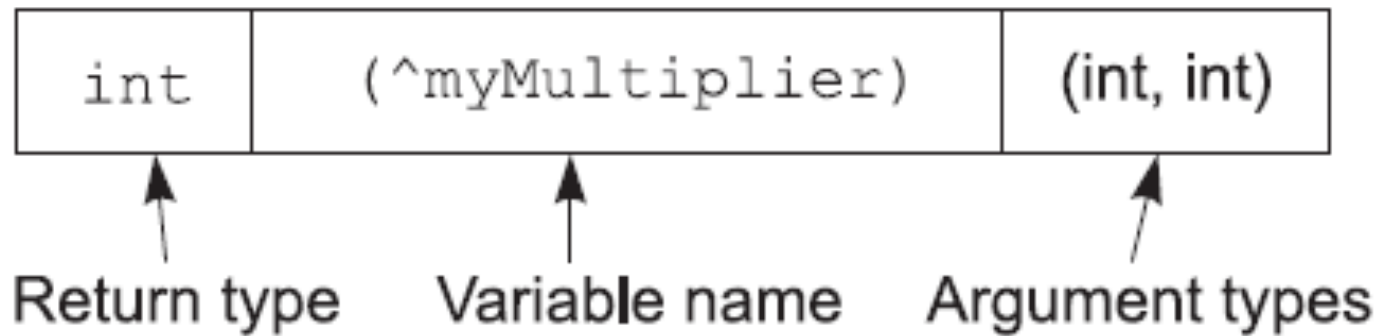
The listing has two parts:

- The variable declaration.
- The block literal (the part after the equals sign).



Variable declaration

- The block name is enclosed in parentheses and has to be introduced by a caret (^)



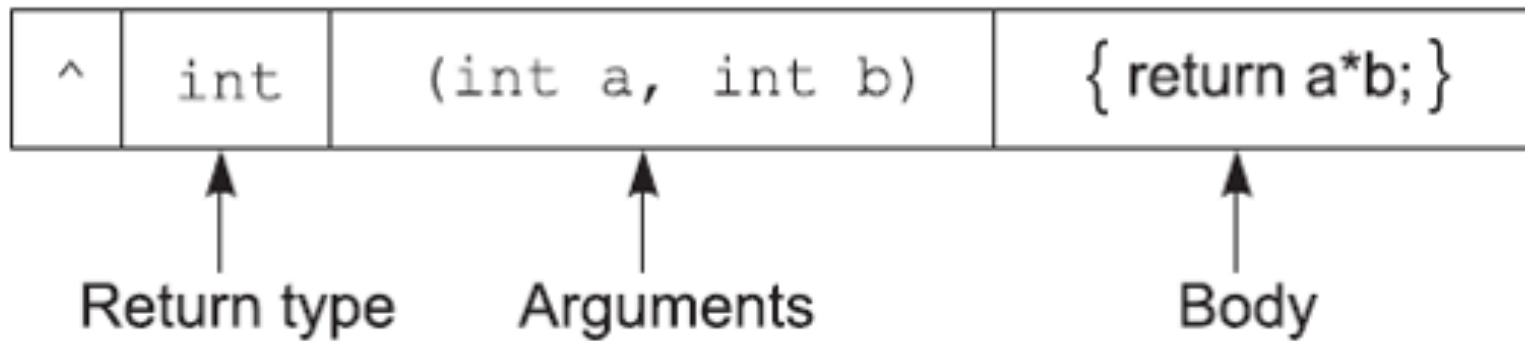
Variable declaration Cont.

```
int (^myMultiplier) (int, int);  
  
myMultiplier = ^int (int a, int b){  
    return a * b;  
};
```



Block literal

- A block literal always starts with the caret (^)



Block literal Cont.

- Both the return type and the arguments are optional.
- If the return type / arguments are void, you can omit them.
- The return type is also optional if it can be predicted automatically by the compiler through the return statement in the block's body.



Usage

```
int (^myMultiplier) (int, int) = ^int (int a, int b){  
    NSLog(@"From inside block");  
    return a * b;  
};
```

```
//Running using return int the variable result  
int result = myMultiplier(10,5);  
NSLog(@"Result is: %d", result);
```

```
//Running without getting return  
myMultiplier(2,4);
```

From inside block
Result is: 50

From inside block



Animated Effects Using UIKit Dynamics



UIKit Dynamics

- UIKit Dynamics is a brand new library shipped in **iOS 7**.
- UIKit Dynamics is part of the UIKit framework, which means that you don't need to add any additional frameworks to your projects to use it.
- It provides developers with an interface to add realistic effects to the view layer of your applications without having deep knowledge of physics, math and of course libraries or frameworks like Core Animation.



UIDynamicAnimator

- **UIDynamicAnimator** is the heart of this library that hides a physics engine, implemented and built right into the UIKit framework.
- It is responsible for producing the animation that will attach all the desired realistic effects to an application.
- Even though the **UIDynamicAnimator** class is the core of **the UIKit Dynamics**, it doesn't do anything on its own.
- Objects of some other classes must be added to it, named behaviors, or programmatically speaking, **UIDynamicBehaviors**.



UIDynamicBehaviors

- A UIKit Dynamics behavior actually represents a physical behavior of the real world in the programming world, and it specifies the realistic effects provided by UIKit Dynamics to developers.
- The following classes are related to behaviors.



UIDynamicBehaviors Cont.

- **UIGravityBehavior**: It's name says it all. The *UIView* objects that is applied to gain gravity attributes.
- **UICollisionBehavior**: It provides the ability to two or more view objects to collide, or between a view object and a boundary (such as the screen bounds).



UIDynamicBehaviors Cont.

- **UIPushBehavior**: It applies a force (in other words an acceleration) towards any direction to one or more view objects, and it supports two modes: The *continuous* where the pushed object gets its speed gradually, and the *instantaneous* where the object is pushed at max speed at once.
- **UIAttachmentBehavior**: It is used to attach a view object to another, or a view object to a point.



UIDynamicBehaviors Cont.

- **UISnapBehavior**: Using this class, a view object can animate like it is *magnetized* from an invisible magnet existing to a specific point. More than that, it contains a *damping* property, which can be used to make the object bounce around the point that it will finally snap to.
- Besides the above, there is also one more important class, the **UIDynamicItemBehavior**. This class doesn't attach a specific behavior to an object, but it's used to adjust certain properties regarding other behaviors.



UIDynamicItemBehavior properties

- **elasticity**: It's value ranges from 0.0 to 1.0 and it specifies how elastic a collision between two objects or an object and a boundary will be.
- **density**: This property represents the mass of an object. The greater its value, the bigger the object's mass.
- **resistance**: Using this property, the damping of an object's velocity can be modified.
- **friction**: With this one, the friction between two view objects can be set.



UIDynamicItemBehavior properties Cont.

- **angularResistance**: It's used for damping the angular velocity of an object.
- **allowsRotation**: This BOOL property can be used to specify if an object should be rotated or not.





Demo

Bouncing Ball



Bouncing Ball demo



1. In the private interface section we add the following object declarations:

```
@interface JETSTFirstViewController : UIViewController  
  
@property (nonatomic , strong) UIDynamicAnimator *animator;  
  
@property (nonatomic , strong) UIView *orangBall;  
  
@end
```



2. Go to the **viewDidLoad** method to initialize our two objects.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    // Setup the ball view.
    self.orangBall = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0, 50.0, 50.0)];
    self.orangBall.backgroundColor = [UIColor orangeColor];
    self.orangBall.layer.cornerRadius = 25.0;
    self.orangBall.layer.borderColor = [UIColor blackColor].CGColor;
    self.orangBall.layer.borderWidth = 0.0;

    [self.view addSubview:self.orangBall];

    // Initialize the animator.
    self.animator = [[UIDynamicAnimator alloc] initWithReferenceView:self.view];
}
```



3. let's declare a private method in which we'll do all of our job:

```
@interface JETSTFirstViewController : UIViewController

@property (nonatomic , strong) UIDynamicAnimator *animator;
@property (nonatomic , strong) UIView *orangBall;

-(void) demoGravity;

@end
```



4. Implement the demoGravity method using the following code.

```
-(void)demoGravity{  
  
    UIGravityBehavior *gravity = [[UIGravityBehavior alloc] initWithItems:@[self.orangBall]];  
    [self.animator addBehavior:gravity];  
  
}
```



5. This step is necessary if we want to see it working.

Go first to the *viewDidLoad* method and invoke this method:

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    ....

    [self demoGravity];
}
```



Bouncing Ball demo steps Cont.

- Now you should see something like that



6. This step is to add a collision to prevent the ball to fall endlessly due to gravity.

```

-(void)demoGravity{

    ....
    ....

    UICollisionBehavior *collisionBehavior = [[UICollisionBehavior alloc] initWithItems:@[self.orangBall]]
    ;

    [collisionBehavior addBoundaryWithIdentifier:@"tabBar" fromPoint:self.tabBarController.tabBar.frame.
        origin toPoint:CGPointMake(self.tabBarController.tabBar.frame.origin.x + self.tabBarController.
        tabBar.frame.size.width, self.tabBarController.tabBar.frame.origin.y)];

    [self.animator addBehavior:collisionBehavior];

}
  
```



Bouncing Ball demo steps Cont.

- Now you can run and you should to see something like that.



- Great, we've created a collision behavior and our ball view doesn't get lost in chaos, however the bouncing doesn't say much yet.
- We can fix this by changing the *elasticity* of the collision, but how can this be done?
- There is a class named *UIDynamicItemBehavior*, which can be used to alter various properties of all the other behaviors.



7. We are going to create an object of this class, and then through it we'll modify the elasticity value. In the *demoGravity* method, add the following code:

```
UIDynamicItemBehavior *ballBehavior = [[UIDynamicItemBehavior alloc] initWithItems:@[self.orangBall]];

ballBehavior.elasticity = 0.75;

[self.animator addBehavior:ballBehavior];
```



Bouncing Ball demo steps Cont.

- Running the app now will show something like the next one.



CocoaPods



CocoaPods

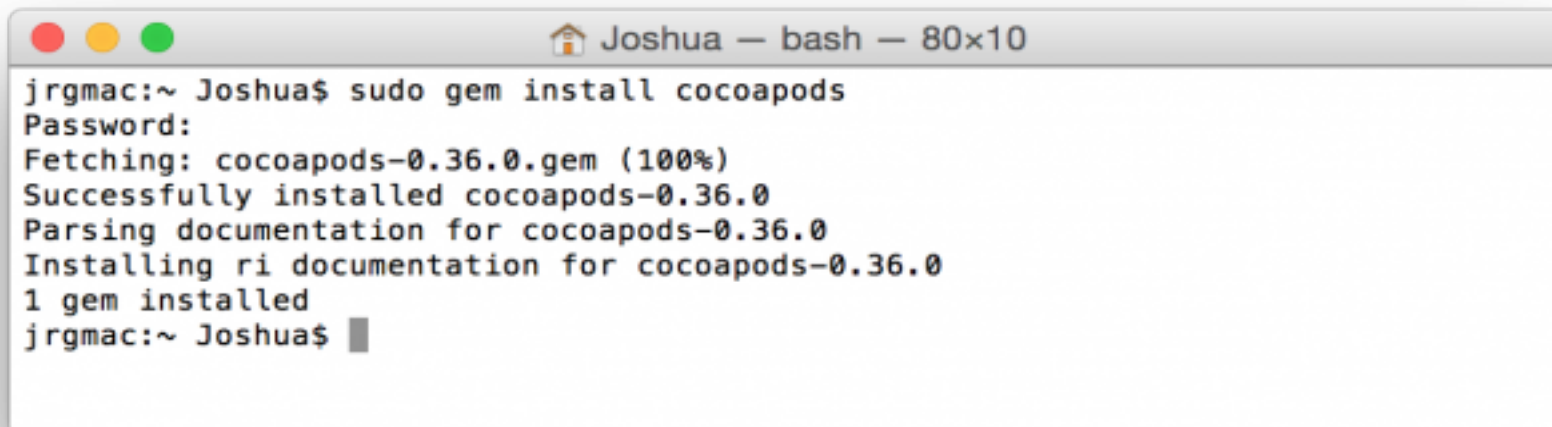
- CocoaPods is a dependency manager for Swift and Objective-C Cocoa projects.
- It has over 27 thousand libraries and is used in over 1.7 million apps.
- CocoaPods can help you scale your projects elegantly.
- CocoaPods is built with Ruby and is installable with the default Ruby available on OS X.



Installation

- CocoaPods is built on Ruby, which ships with all recent versions of Mac OS X. This has been the case since OS X 10.7.
- To install CocoaPods open Terminal and enter the following command:

```
sudo gem install cocoapods
```



```
Joshua — bash — 80x10
jrgmac:~ Joshua$ sudo gem install cocoapods
Password:
Fetching: cocoapods-0.36.0.gem (100%)
Successfully installed cocoapods-0.36.0
Parsing documentation for cocoapods-0.36.0
Installing ri documentation for cocoapods-0.36.0
1 gem installed
jrgmac:~ Joshua$
```



Installation Cont.

- Enter this command in Terminal to complete the setup:

```
pod setup --verbose
```

- This process will likely take a few minutes as it clones the [CocoaPods Master Specs repository](#) into `~/cocoapods/` on your computer.
- The verbose option logs progress as the process runs, allowing you to watch the process instead of seeing a seemingly “frozen” screen.



Installing Your First Dependency

- You first need to close **Xcode**.
- Open **Terminal** and navigate to the directory that contains your project.

```
cd ~/Path/To/Folder/Containing/IceCreamShop
```



Installing Your First Dependency Cont.

- Next, enter this command:

```
pod init
```

- This creates a [Podfile](#) for your project.
- Type this command to open the Podfile using Xcode for editing:

```
open -a Xcode Podfile
```



Installing Your First Dependency Cont.

```
platform :ios, '9.0'  
  
target 'TheNameOfYourProject' do  
  
  pod 'SDWebImage', '~>3.8'  
  
end
```

- After add the pod command in the Podfile you now need to tell CocoaPods to install the dependencies for your project.
- Enter the following command in ***Terminal***.

`pod install`



Cocoapods Demo



Lab Exercise



1. Bouncing Ball

- Create Bouncing Ball like that:



- **Note:** After the bouncing the color of the ball should be changed use : (**UICollisionBehaviorDelegate**).



2. Show Images using SDWebImage

- Create UITableView that shows rows of images.
- Images will be retrieved from the following URL: https://images.rawpixel.com/image_png_1300/czNmcy1wcml2YXRlL3Jhd3BpeGVsX2ltYWdlcy93ZWJzaXRlX2Nvb3RlbnQvbHlvcml00NTAtMjMucG5n.png
- Retrieve images using SDWebImage using Cocoapods.

