

Chapter 4

Views and View Hierarchy

- Views and View Hierarchy
- Creating custom views

View Basics

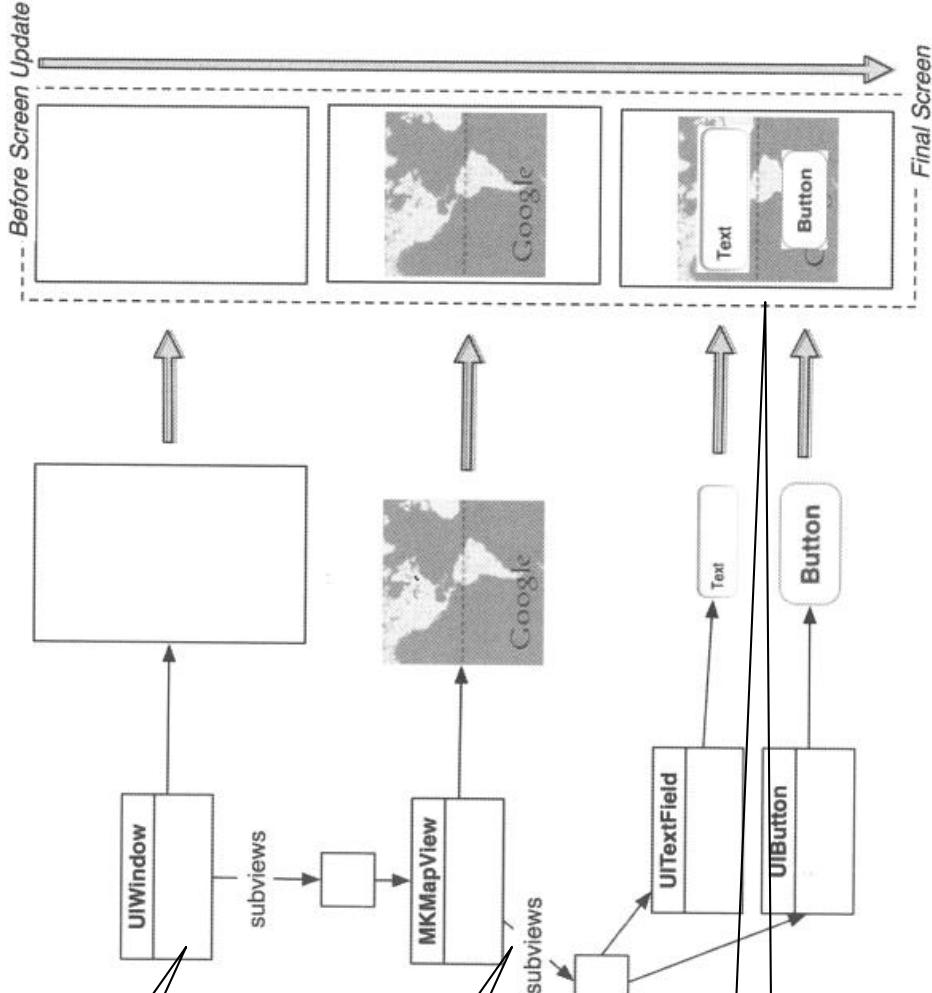
- A view is an instance of UIView or one of its subclass
- A view knows how to draw itself
- A view handles events, like touches and gestures
- A view exists within a hierarchy of views
- Views may be added to the application by the designer, similar to what we did in the Quiz application.
- Views may be generated programmatically by the application when needed.

Views and View Hierarchy

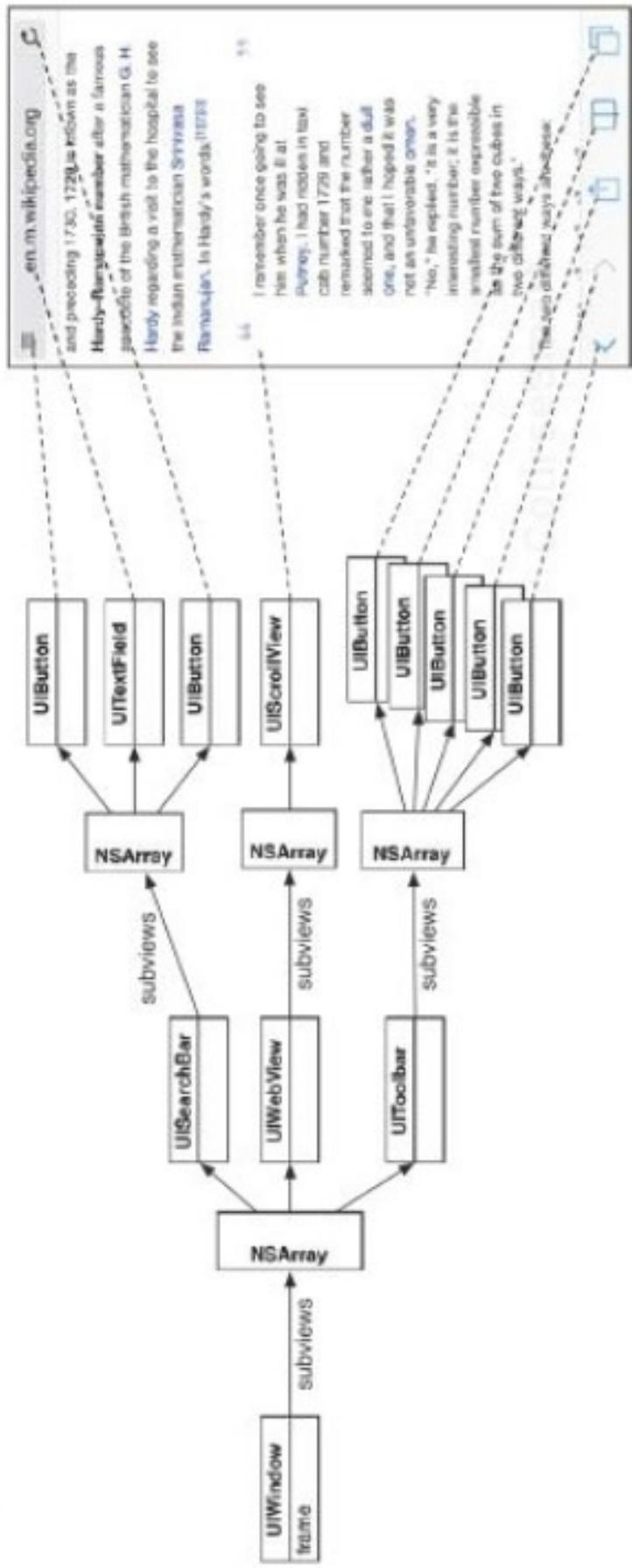
A cocoa touch application starts with a main window which is an instance of UIWindow

Subviews are layered on top of the main window to create the user interface.

Each subview will draw its own image and add it to the views hierarchy.

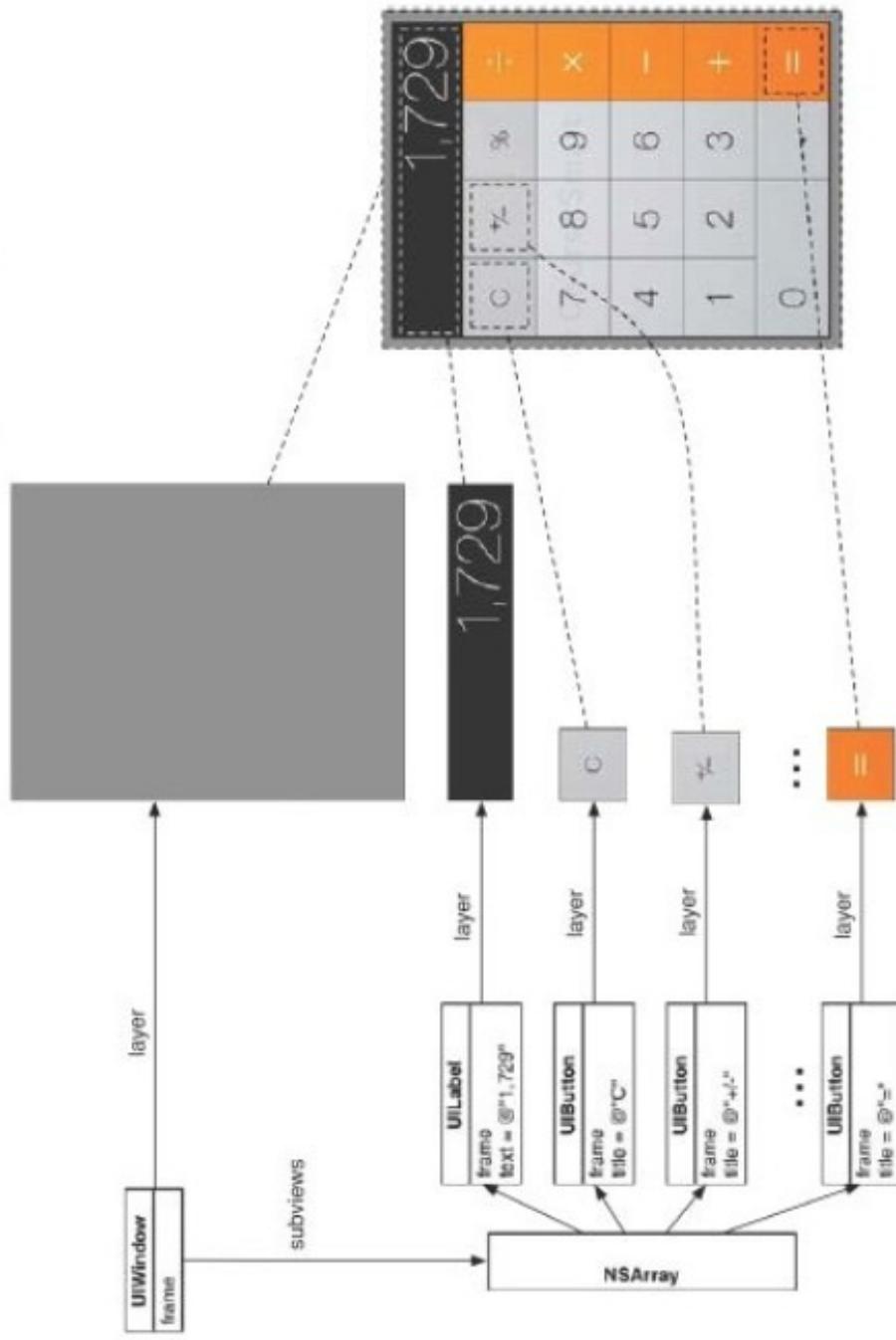


Sample View Hierarchy

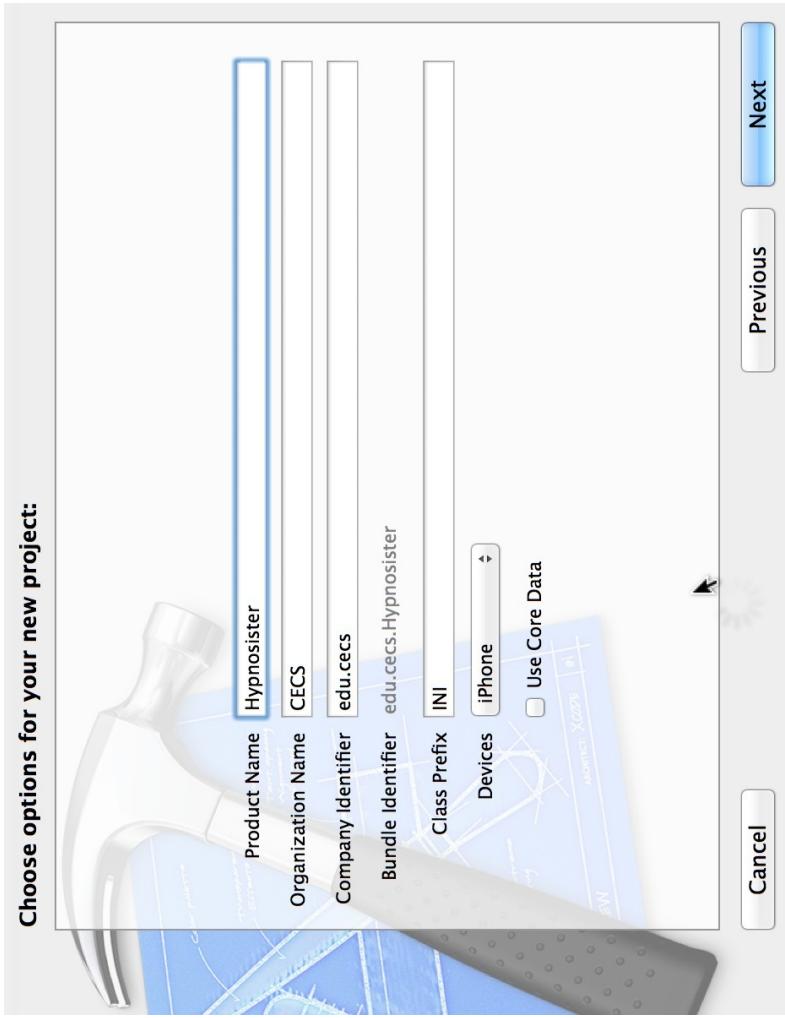


- Each view in the hierarchy, including the window, draws itself. It renders itself to its layer, which is an instance of CALayer. (You can think of a view's layer as a bitmap image.)
 - The layers of all the views are composited together on the screen.

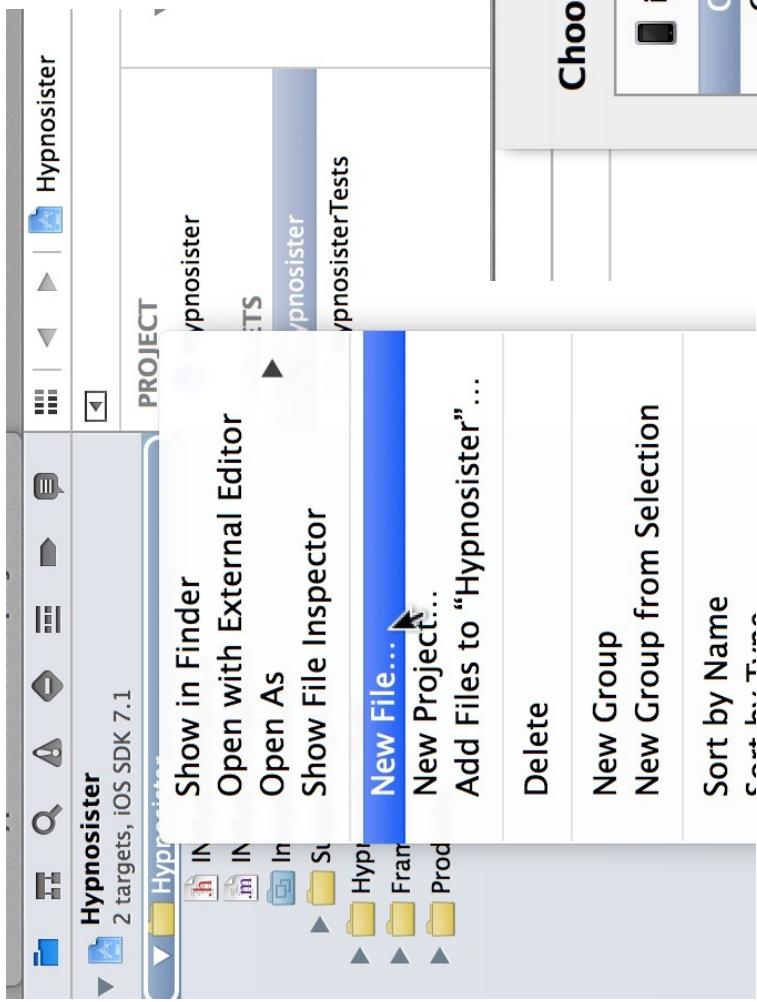
- Views render themselves and then are composited together



Creating Hypnosister

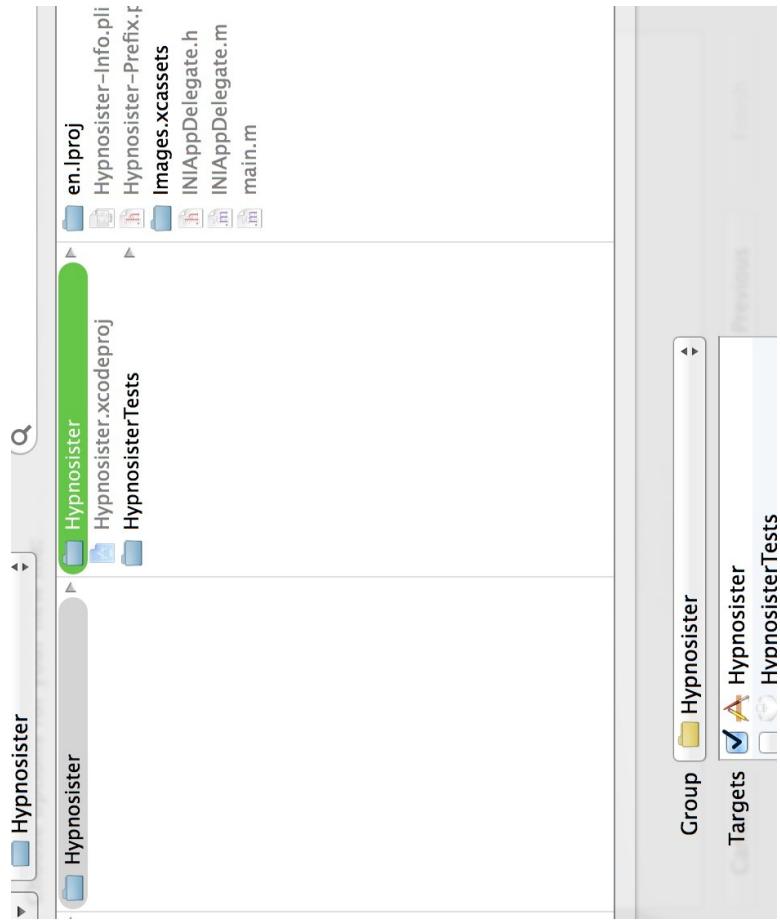
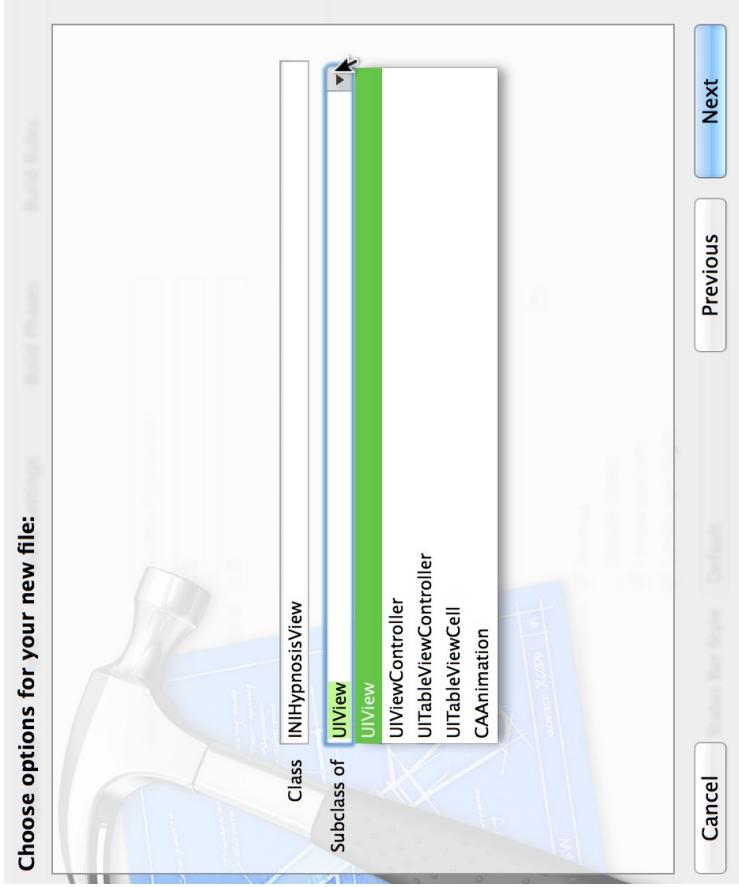


Creating NI HypnosisView



Choose a template for your new file:





Views and Frames

INIHypnosisView.m @implementation INIHypno-

```
1 // INIHypnosisView.m
2 // Hypnosister > Hypnosister > INIHypnosisView.m > @implementation INIHypno-
3 // Hypnosister
4 // Created by Ibrahim Imam on 5/19/14.
5 // Copyright (c) 2014 CECS. All rights reserved.
6 //
7 //

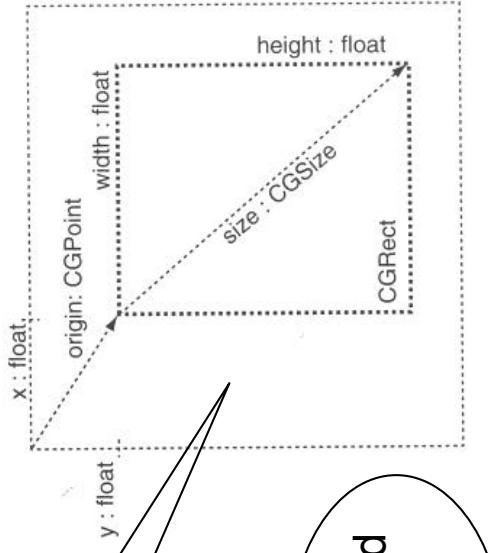
8 #import "INIHypnosisView.h"
9
10 @implementation INIHypnosisView
11
12 - (id)initWithFrame:(CGRect)frame
13 {
14     self = [super initWithFrame:frame];
15     if (self) {
16         // Initialization code
17         // Drawing code
18     }
19     return self;
20 }

21 /*
22 // Only override drawRect: if you perform custom drawing.
23 // An empty implementation adversely affects performance during animation.
24 - (void)drawRect:(CGRect)rect
25 {
26     // Drawing code
27 }
28 */
29
30 @end
31
32
```

CGRect

CGRect

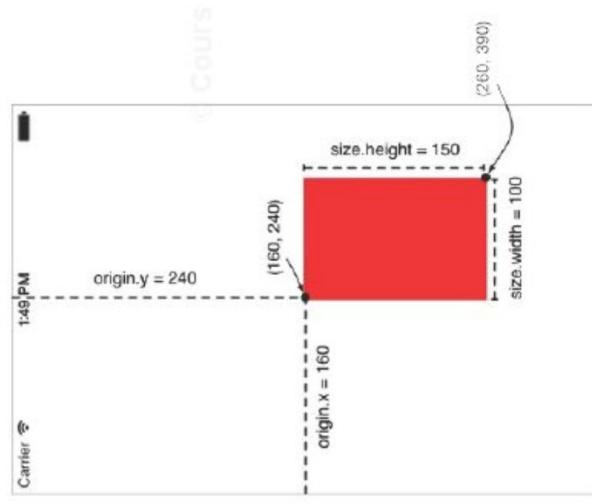
Designated
Initializer



- A CGRect contains the members origin and size.
- The origin is a C structure of type CGPoint and contains two float members: x and y.
- The size is a C structure of type CGSize and has two float members: width and height

Creating the main window and CGRect

- A view's frame specifies the view's size and its position relative to its superview.
- A view's size is always specified by its frame and a view is always a rectangle.
- The AppDelegate.m usually implements the method "application:didFinishLaunchingWithOptions:" to create the main window of the application.



Making
a Struct

Creating a subview and adding
it to the main window

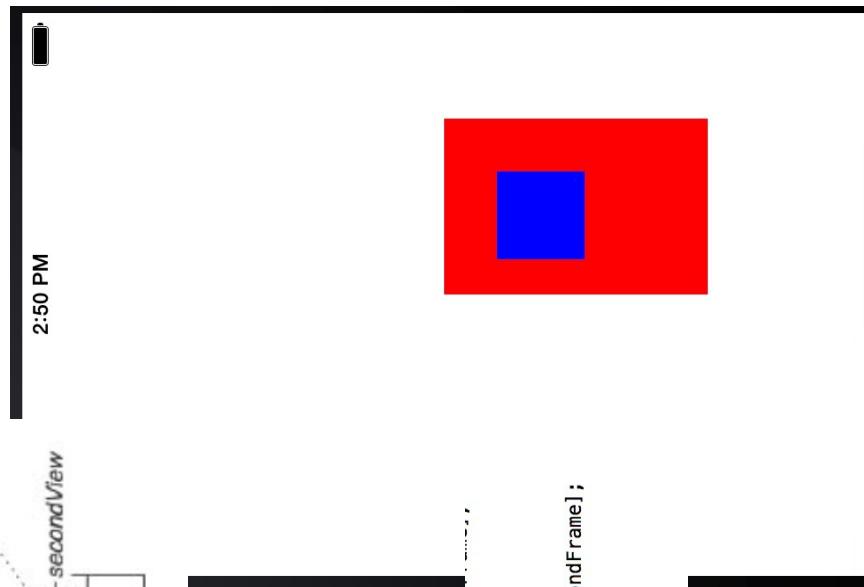
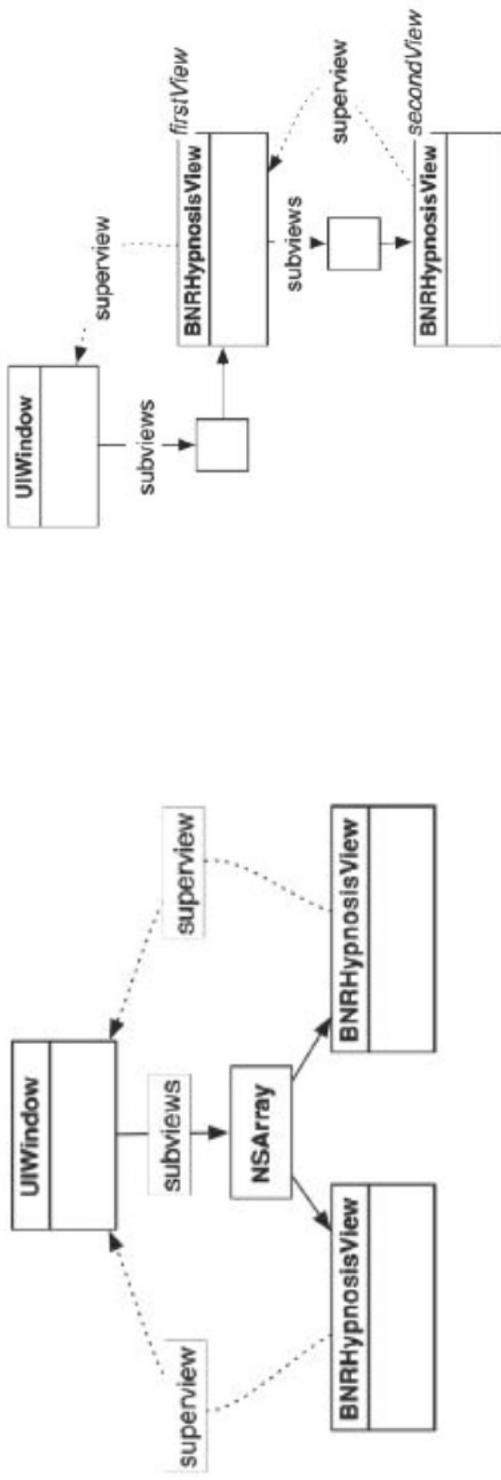
```
13 - (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions
14 {
15     self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds]];
16     CGRect firstFrame = CGRectMake(160, 240, 100, 150);
17     INTHypnosisView *firstView = [[INTHypnosisView alloc] initWithFrame: firstFrame];
18     firstView.backgroundColor = [UIColor redColor];
19     self.window.addSubview(firstView];
20     self.window.backgroundColor = [UIColor whiteColor];
21     self.window.makeKeyAndVisible];
22     return YES;
23 }
```

View Controllers

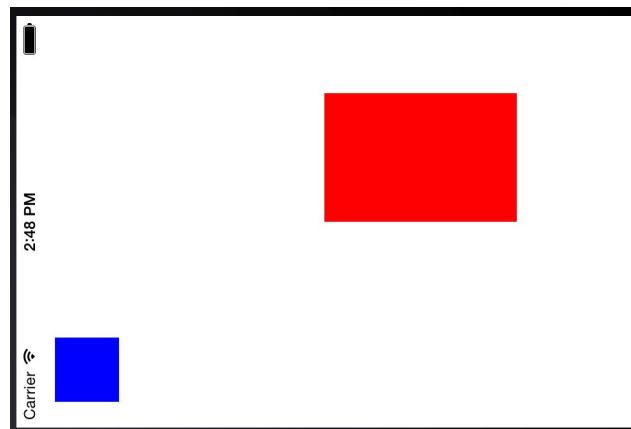
2014-05-25 14:18:04.688 Hypnosister[984:60b] Application windows are expected to have a root view controller at the end of application launch

- A view controller is an object that controls some set of an application's view hierarchy
- Most iOS apps have one or more view controllers.
- Hypnosister is simple enough that it does not need a view controller, so you can ignore this comment.

Adding More Subviews

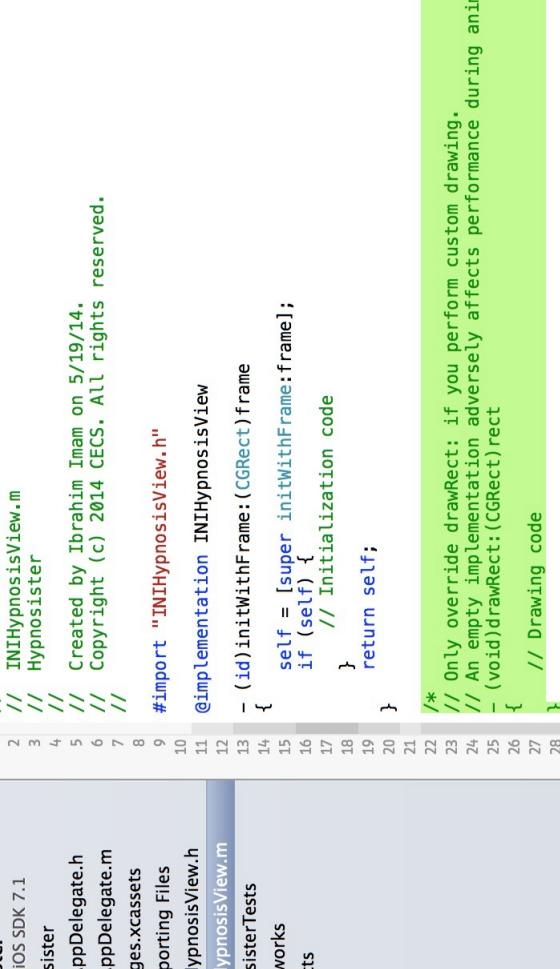


```
firstView.backgroundColor = [UIColor redColor];
[self.window addSubview: firstView];
CGRect secondFrame = CGRectMake( 20, 30, 50, 50);
INIHypnosisView *secondView = [[INIHypnosisView alloc] initWithFrame: secondFrame];
secondView. backgroundColor = [UIColor blueColor];
// [self.window addSubview: secondView];
[firstView addSubview: secondView];
```



Custom Drawing using drawRect: Method

- In UIView subclasses that we create we use the method “drawRect:” to implement code needed to draw our image.
 - “drawRect:” method is the rendering step a view uses to draw itself
 - In the NIHypnosisView we use “drawRect:” to draw a series on concentric circles with a text image in the middle

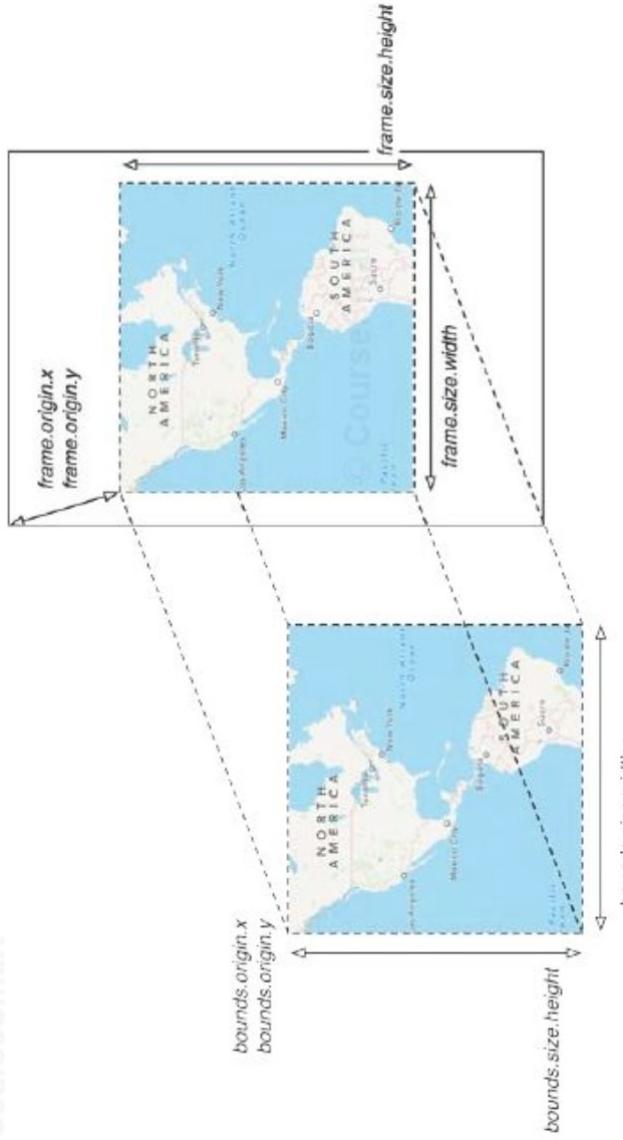


```
1 // INIHypnosisView.m
2 // Hypnosister
3 // Hypnosister
4 // Created by Ibrahim Imam on 5/19/14.
5 // Copyright (c) 2014 CECS. All rights reserved.
6 //
7 //
8 #import "INIHypnosisView.h"
9 @implementation INIHypnosisView
10
11 - (id)initWithFrame:(CGRect)frame
12 {
13     self = [super initWithFrame:frame];
14     if (self) {
15         // Initialization code
16     }
17     // Drawing code
18 }
19
20 return self;
21
22 /*
23 // Only override drawRect: if you perform custom drawing.
24 // An empty implementation adversely affects performance during animation.
25 - (void)drawRect:(CGRect)rect
26 {
27     // Drawing code
28 }
29 */
30
31 @end
```

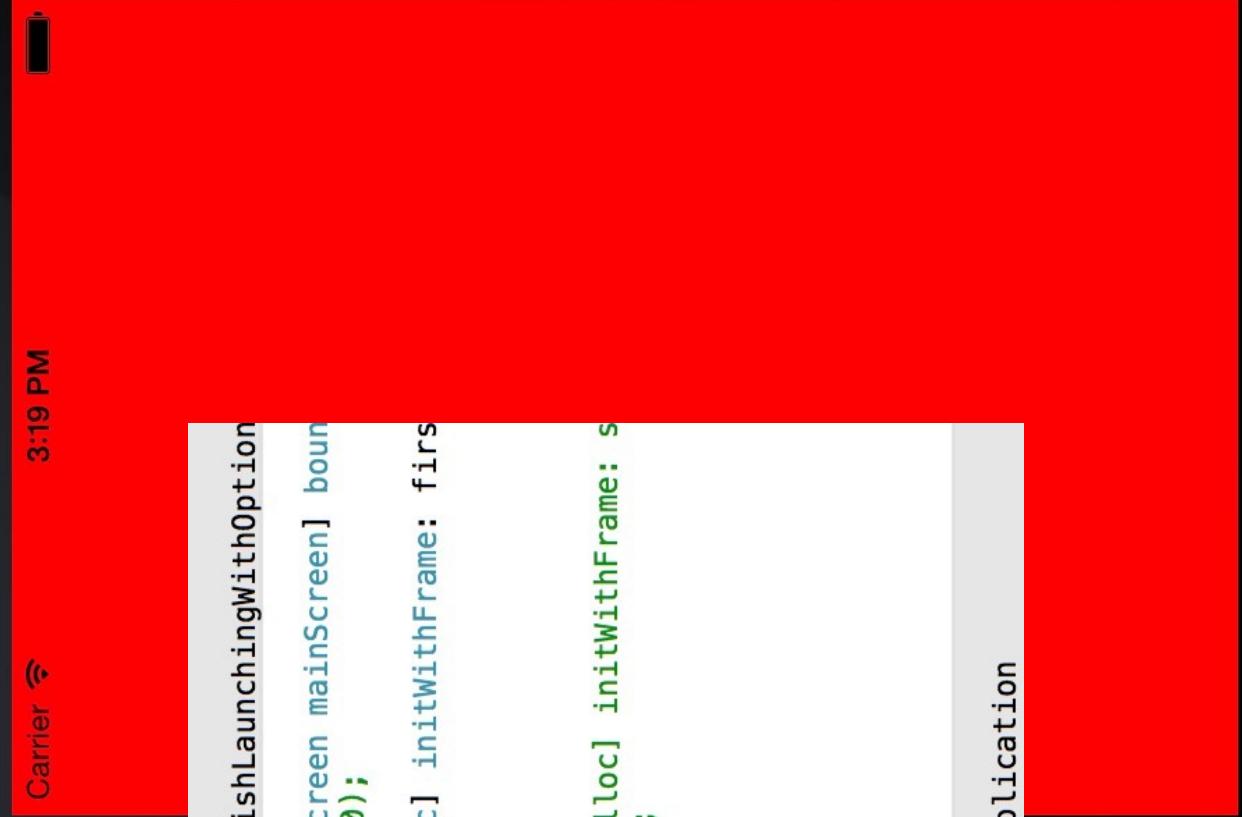
Defining
Hypnosis View

bounds vs. frame

- The first thing that you typically do when overriding `drawRect:` is get the bounds rectangle of the view.
- The frame and bounds rectangles have distinct purposes.
 - A view's frame rectangle is used during compositing to lay out the view's layer relative to the rest of the view hierarchy.
 - The bounds rectangle is used during the rendering step to lay out detailed drawing within the boundaries of the view's layer.



Have our view fill the screen



The screenshot shows a portion of an Xcode code editor. The status bar at the top indicates "Carrier" and "3:19 PM". The main area contains Objective-C code:

```
- (BOOL)application:(UIApplication *)application didFinishLaunchingWithOptions:(NSDictionary *)launchOptions {
    self.window = [[UIWindow alloc] initWithFrame:[UIScreen mainScreen] bounds];
    // CGRect firstFrame = CGRectMake(160, 240, 100, 150);
    CGContext firstFrame = self.window.bounds;
    INIHypnosisView *firstView = [[INIHypnosisView alloc] initWithFrame: firstFrame];
    firstView.backgroundColor = [UIColor redColor];
    [self.window addSubview: firstView];

    // CGRect secondFrame = CGRectMake(20, 30, 50, 50);
    INIHypnosisView *secondView = [[INIHypnosisView alloc] initWithFrame: secondFrame];
    secondView.backgroundColor = [UIColor blueColor];

    [self.window addSubview: secondView];
    [firstView addSubview: secondView];

    self.window.backgroundColor = [UIColor whiteColor];
    [self.window makeKeyAndVisible];
    return YES;
}

- (void)applicationWillResignActive:(UIApplication *)application
```

The code is annotated with green highlights: "self.window" and "firstFrame" are highlighted in green, indicating they are selected or being edited.

Getting Help

```
23 // Only override drawRect: if you perform custom drawing.
24 // An empty implementation adversely affects performance during animation.
25
26 - (void)drawRect:(CGRect)rect
27 {
28     CGRect bounds = self.bounds;
29     //Figureoutthecenteroftheboundsrectangle
30
31     CGPoint center;
32     center.x = bounds.origin.x + bounds.size.width / 2.0;
33     center.y = bounds.origin.y + bounds.size.height / 2.0;
34
35     // The circle will
36     float radius = MIN(
37         UIBezierPath.path =
38
39     }
```

<p>Description The UIBezierPath class lets you define a path consisting of straight and curved line segments and render that path in your custom views. You use this class initially to specify just the geometry for your path. Paths can define simple shapes such as rectangles, ovals, and arcs or they can define complex polygons that incorporate a mixture of straight and curved line segments. After defining the shape, you can use additional methods of this class to render the path in the current drawing context.</p> <p>Availability iOS (3.2 and later)</p> <p>Declared In UIBezierPath.h</p> <p>Reference UIBezierPath Class Reference</p>	<p>Description The UIBezierPath class lets you define a path consisting of straight and curved line segments and render that path in your custom views. You use this class initially to specify just the geometry for your path. Paths can define simple shapes such as rectangles, ovals, and arcs or they can define complex polygons that incorporate a mixture of straight and curved line segments. After defining the shape, you can use additional methods of this class to render the path in the current drawing context.</p> <p>Availability iOS (3.2 and later)</p> <p>Declared In UIBezierPath.h</p> <p>Reference UIBezierPath Class Reference</p>	<p>Availability iOS (3.2 and later)</p> <p>Declared In UIBezierPath.h</p> <p>Reference UIBezierPath Class Reference</p>
<p>Guides Drawing and Printing Guide for iOS</p> <p>Sample Code HeadsUpUI, MapCallouts, MoveMe, WiTap, iAdInterstitialSuite</p>	<p>Guides Drawing and Printing Guide for iOS</p> <p>Sample Code HeadsUpUI, MapCallouts, MoveMe, WiTap, iAdInterstitialSuite</p>	<p>Guides Drawing and Printing Guide for iOS</p> <p>Sample Code HeadsUpUI, MapCallouts, MoveMe, WiTap, iAdInterstitialSuite</p>

Classes and API Documentation

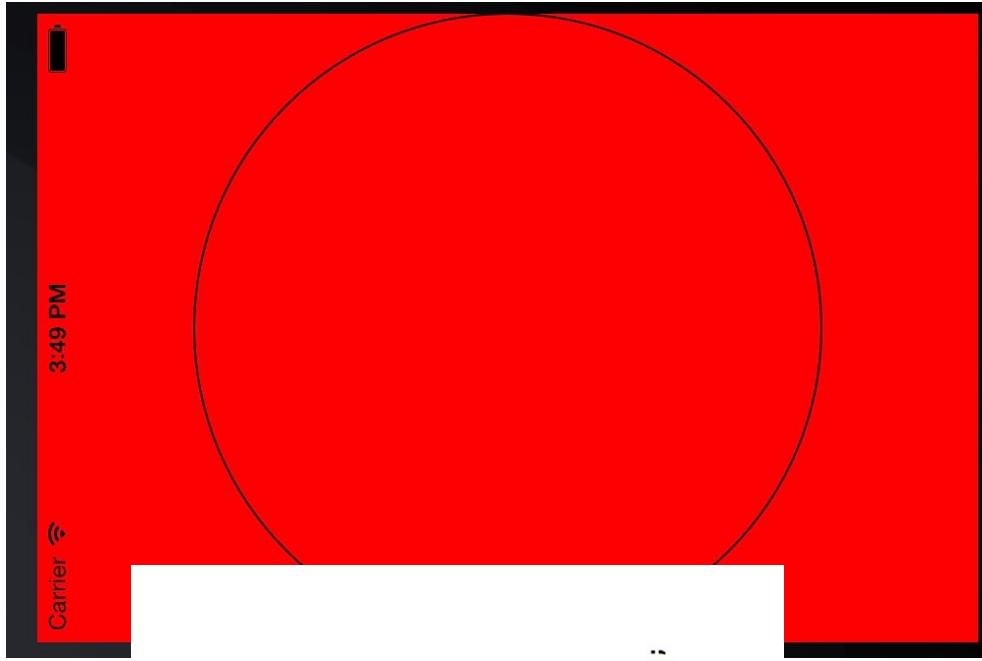
The screenshot shows the UIKit documentation for the `UIBezierPath` class. The main content area is divided into several sections:

- Overview**: A brief description of what `UIBezierPath` does.
- Inheritance**: Shows that `UIBezierPath` inherits from `NSObject` and conforms to `NSCopying` and `NSCoding`. It also lists the `UIKit` framework.
- Methods**: A large section listing methods such as `moveToPoint:`, `addLineToPoint:`, `addArcWithCenter:radius:startAngle:endAngle:clockwise:`, `addCurveToPoint:controlPoint1:controlPoint2:`, `addQuadCurveToPoint:controlPoint:`, `closePath`, `removeAllPoints`, `appendPath:`, `CGPath`, `property`, `currentPoint`, `property`, `currentPoint`, `property`, `lineWidth`, and `miterLimit`.
- Properties**: A list of properties including `bounds`, `CGPath`, `currentPoint`, `empty`, `flatness`, `lineCapStyle`, `lineJoinStyle`, `lineWidth`, and `miterLimit`.
- Constructing a Path**: A section describing how to construct a path using various methods.
- Accessing Drawing Properties**: A section describing properties related to drawing.
- Clipping Paths**: A section describing clipping paths.
- Hit Detection**: A section describing hit detection.
- Applying Transformations**: A section describing applying transformations.
- Properties**: A list of properties including `bounds`, `CGPath`, `currentPoint`, `empty`, `flatness`, `lineCapStyle`, `lineJoinStyle`, `lineWidth`, and `miterLimit`.

Overriding the drawRect: Method

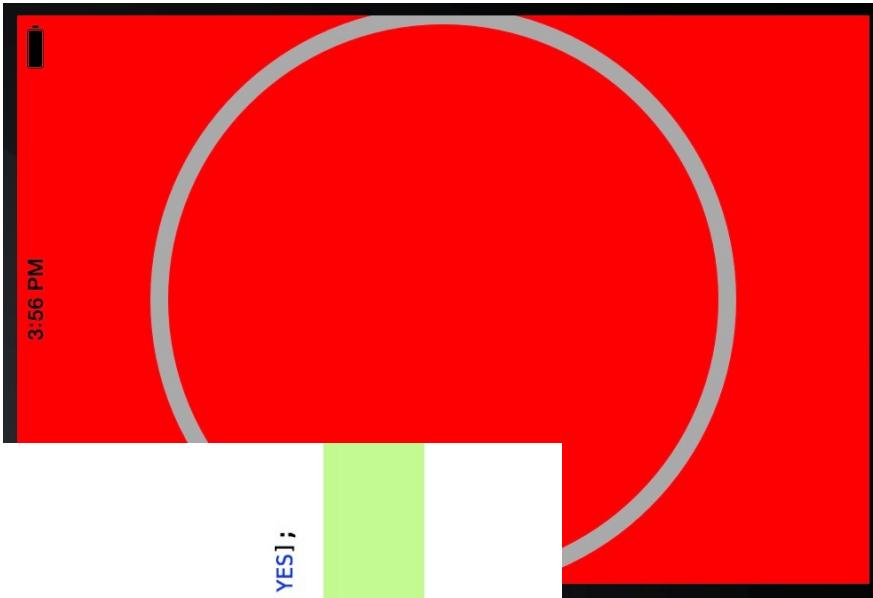
Using the drawRect: method to draw our circles

```
// Only override drawRect: if you perform custom drawing.  
// An empty implementation adversely affects performance during animation.  
- (void)drawRect:(CGRect)rect  
{  
    CGRect bounds = self.bounds;  
    //Figure out the center of the bounds rectangle  
  
    CGPoint center;  
    center.x = bounds.origin.x + bounds.size.width / 2.0;  
    center.y = bounds.origin.y + bounds.size.height / 2.0;  
  
    // The circle will be the largest that will fit in the view  
    float radius = MIN( bounds.size.width, bounds.size.height ) / 2.0;  
  
    UIBezierPath *path = [[ UIBezierPath alloc] init];  
  
    // Add an arc to the path at center, with radius of radius,  
    // from 0 to 2* PI radians ( a circle)  
  
    path addArcWithCenter: center radius: radius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES;  
  
    // Draw the line!  
    [path stroke];  
}
```



Set Stroke Color and Thickness

```
22 // Only override drawRect: if you perform custom drawing.
23 // An empty implementation adversely affects performance during animation.
24 -
25 - (void)drawRect:(CGRect)rect
26 {
27     CGRect bounds = self.bounds;
28     //Figure out the center of the bounds rectangle
29
30     CGPoint center;
31     center.x = bounds.origin.x + bounds.size.width / 2.0;
32     center.y = bounds.origin.y + bounds.size.height / 2.0;
33
34     // The circle will be the largest that will fit in the view
35     float radius = MIN( bounds.size.width, bounds.size.height ) / 2.0;
36
37     UIBezierPath *path = [[ UIBezierPath alloc] init];
38
39     // Add an arc to the path at center, with radius of radius,
40     // from 0 to 2* PI radians ( a circle )
41
42     [path addArcWithCenter: center radius: radius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES];
43
44     // Configure line width to 10 points
45     path.lineWidth = 10;
46
47     // Configure the drawing color to light gray
48     [[UIColor lightGrayColor] setStroke];
49
50     // Draw the line!
51     [path stroke];
52 }
53 }
```



Drawing Concentric Circles

```
28     - (void)drawRect:(CGRect)rect
29     {
30         CGRect bounds = self.bounds;
31         //Figure out the center of the bounds rectangle
32         CGPoint center;
33         center.x = bounds.origin.x + bounds.size.width / 2.0;
34         center.y = bounds.origin.y + bounds.size.height / 2.0;
35
36         // The circle will be the largest that will fit in the view
37         // float radius = MIN( bounds.size.width, bounds.size.height ) / 2.0;
38
39         float maxRadius = hypot( bounds.size.width, bounds.size.height ) / 2.0;
40
41         UIBezierPath *path = [[UIBezierPath alloc] init];
42
43         // Add an arc to the path at center, with radius of radius,
44         // from 0 to 2* PI radians ( a circle )
45         // [path addArcWithCenter: center radius: radius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES];
46
47         for ( float currentRadius = maxRadius; currentRadius > 0; currentRadius -= 20 )
48         {
49             [path addArcWithCenter: center radius: currentRadius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES];
50         }
51
52
53         // Configure line width to 10 points
54         path.lineWidth = 10;
55         // Configure the drawing color to light gray
56         [UIColor lightGrayColor] setStroke];
57
58         // Draw the line!
59         [path stroke];
60     }
```



Cleaning Up

```
28 - (void)drawRect:(CGRect)rect
29 {
30     CGRect bounds = self.bounds;
31     //Figure out the center of the bounds rectangle
32
33     CGPoint center;
34     center.x = bounds.origin.x + bounds.size.width / 2.0;
35     center.y = bounds.origin.y + bounds.size.height / 2.0;
36
37     // The circle will be the largest that will fit in the view
38     // float radius = MIN( bounds.size.width, bounds.size.height ) / 2.0;
39
40     float maxRadius = hypot( bounds.size.width, bounds.size.height ) / 2.0;
41
42     UIBezierPath *path = [[UIBezierPath alloc] init];
43
44     // Add an arc to the path at center, with radius of radius,
45     // from 0 to 2* PI radians ( a circle )
46     // [path addArcWithCenter: center radius: radius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES];
47
48     for ( float currentRadius = maxRadius; currentRadius > 0; currentRadius -= 20 )
49     {
50         [path moveToPoint: CGPointMake( center.x + currentRadius, center.y )];
51         [path addArcWithCenter: center radius: currentRadius startAngle: 0.0 endAngle: M_PI * 2.0 clockwise: YES];
52     }
53
54     // Configure line width to 10 points
55     path.lineWidth = 10;
56     // Configure the drawing color to light gray
57     [[UIColor lightGrayColor] setStroke];
58
59     // Draw the line!
60     [path stroke];
61
62 }
```

