Views

View Hierarchy

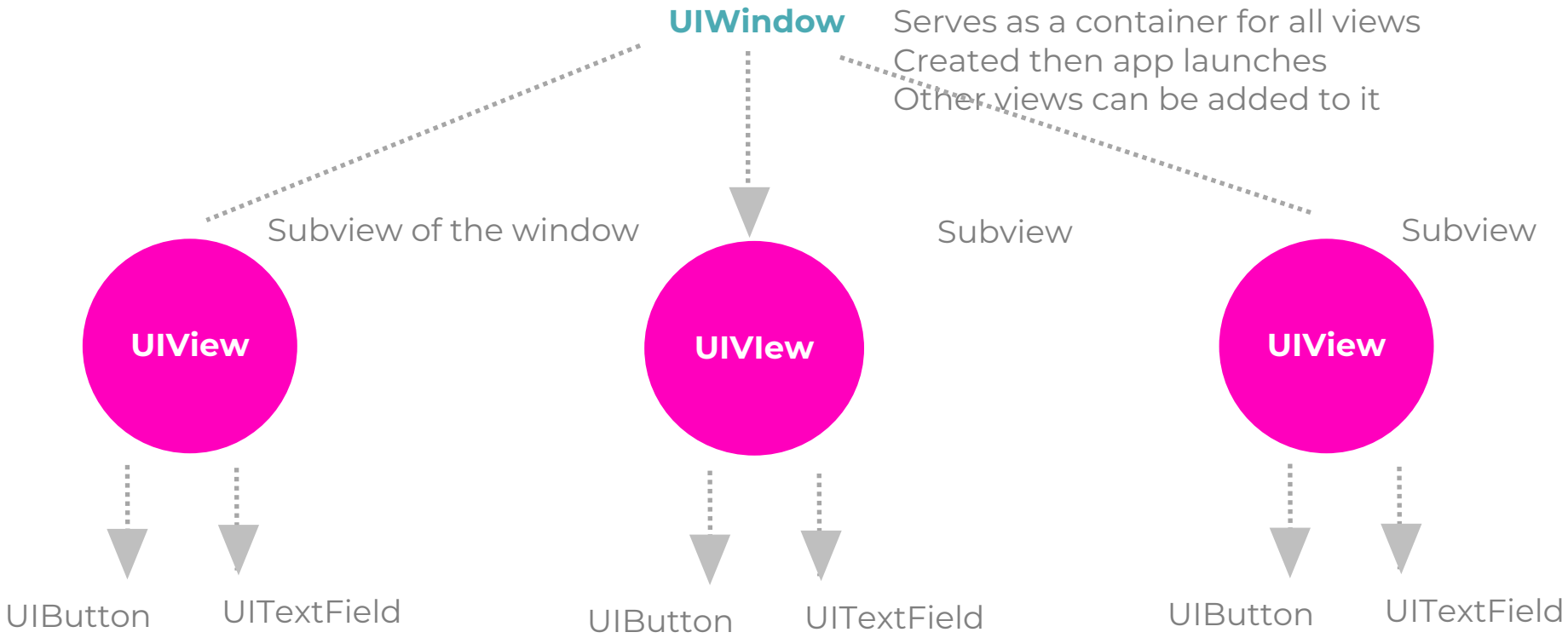
View

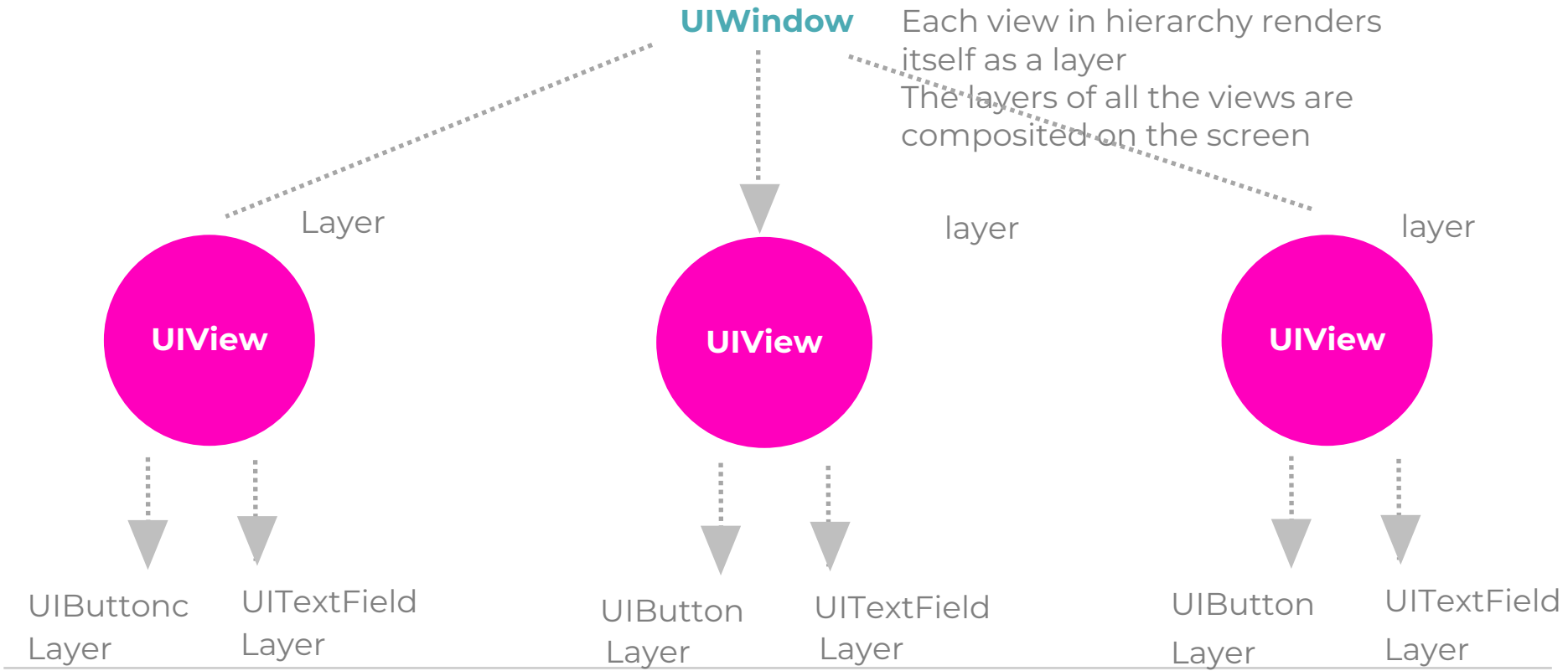
View objects make an app UI

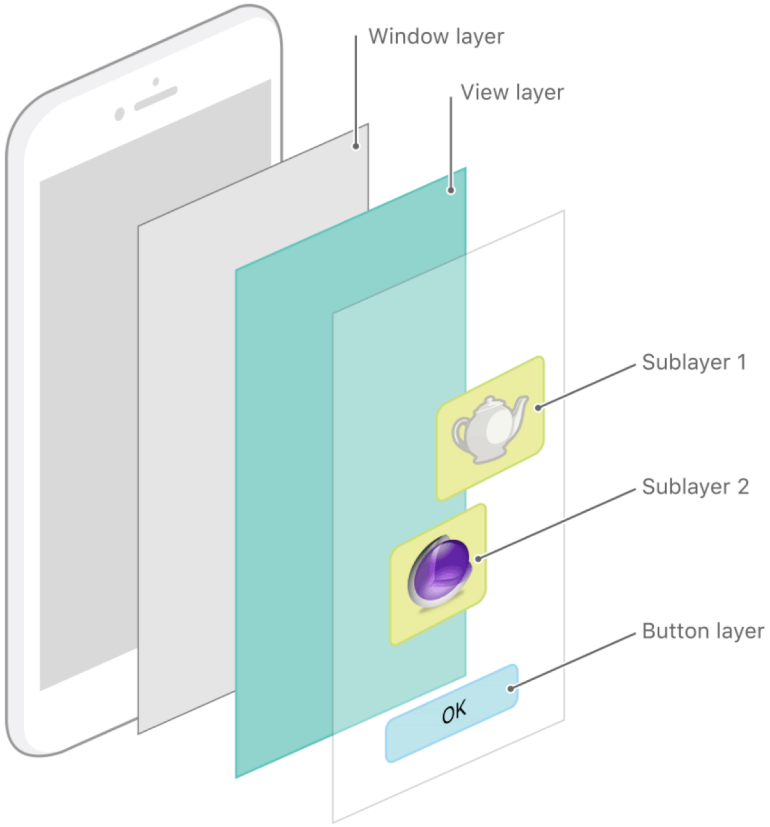
Buttons
Text fields
Sliders

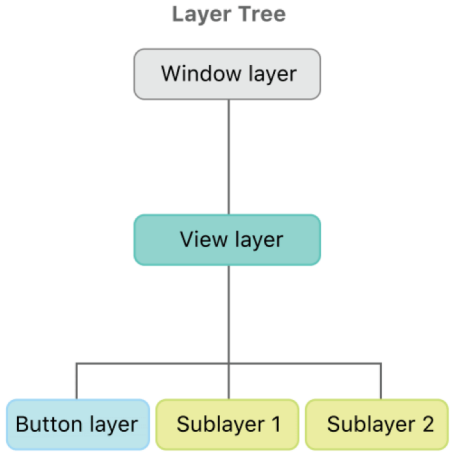
A view

- An instance of UIView
- Knows how to draw itself
- Can handle events (touch)
- Exists within a hierarchy of views whose root is the apps window


UIWindow
Serves as a container for all views
Created then app launches
Other views can be added to it
Subview of the window
Subview
Subview
UIView
UIView
UIView
UIButton
UITextField
UIButton
UITextField
UIButton
UITextField


UIWindow
Each view in hierarchy renders itself as a layer
The layers of all the views are composited on the screen
Layer
layer
layer
UIView
UIView
UIView
UIButtonc Layer
UITextField Layer
UIButton Layer
UITextField Layer
UIButton Layer
UITextField Layer


Window layer
View layer
Sublayer 1
Sublayer 2
Button layer
OK


Layer Tree
Window layer
View layer
Button layer
Sublayer 1
Sublayer 2

World Trotter Example

Single View app
WorldTrotter
Swift
Universal
Include Unit Tests
Include UI Tests

The app which convert values between degrees Fahrenheit and degrees Celsius

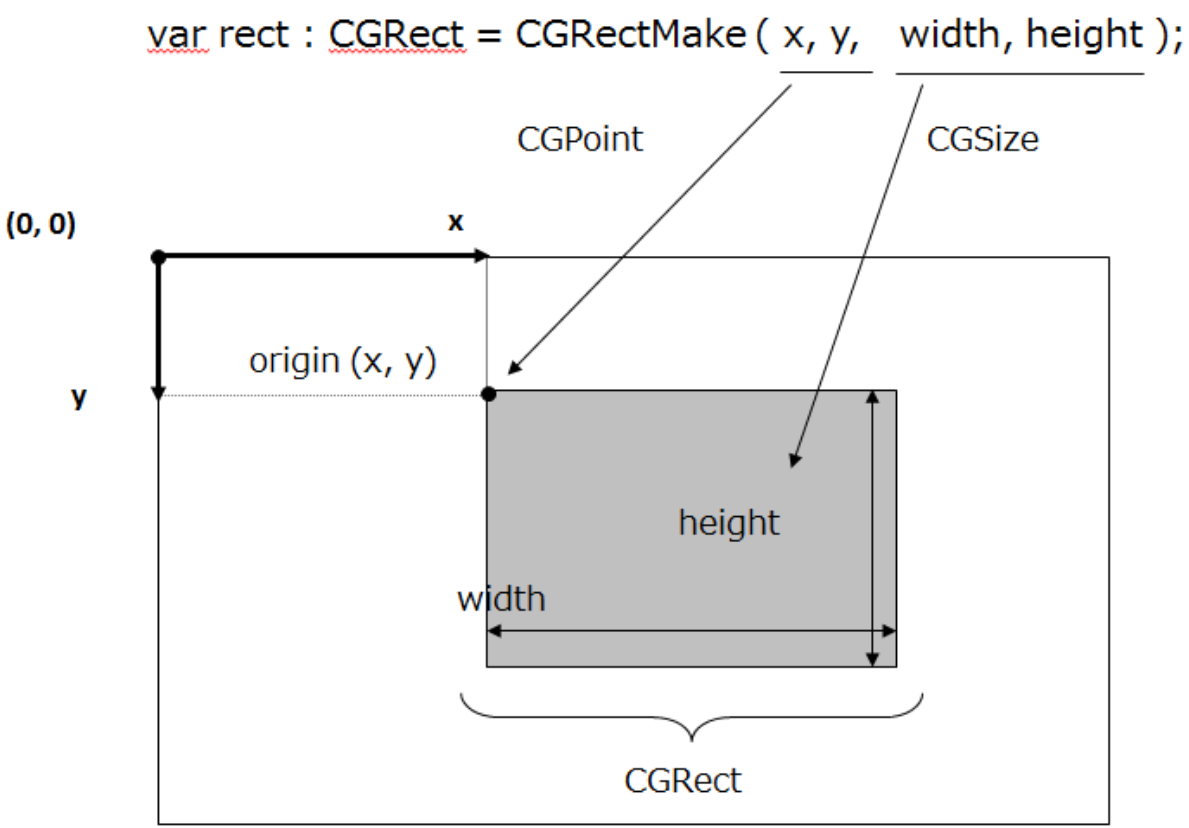
Init (frame)

CGRect – a property of UIView

var frame: CGRect

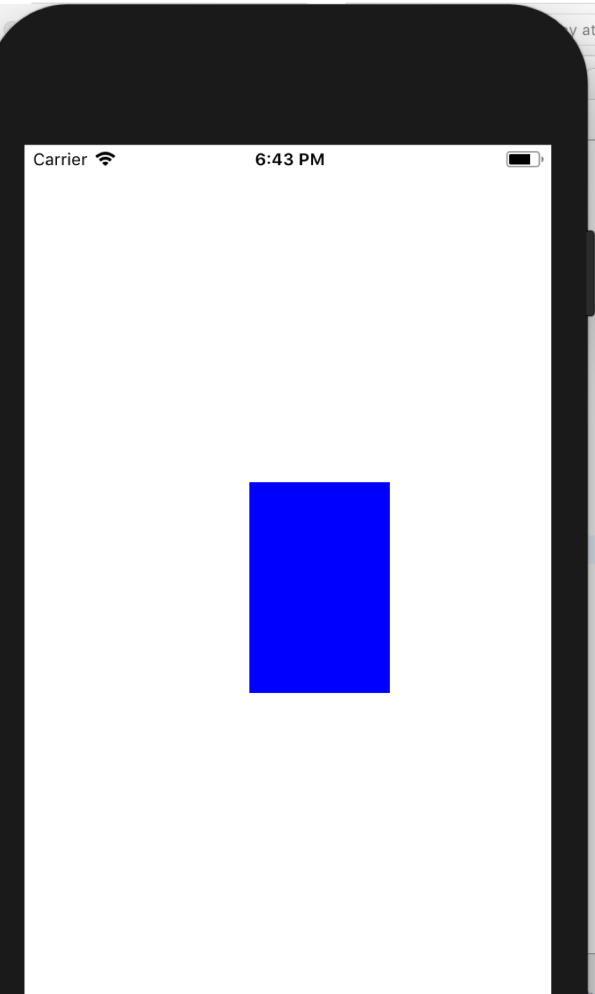
let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)

When app is launched, the view of the initil view controller is added to the root level window. View controller has a view and it is associated with the main view controller for the application is added as a subview



Open ViewController.swift
Delete any methods
Import UIKit

```
import UIKit

class ViewController: UIViewController {

    override func viewDidLoad()
    {
        super.viewDidLoad()

        let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
        let firstView = UIView(frame: firstFrame)
        firstView.backgroundColor = UIColor.blue
        view.addSubview(firstView)
    }
}
```



at 6:43 PM

2

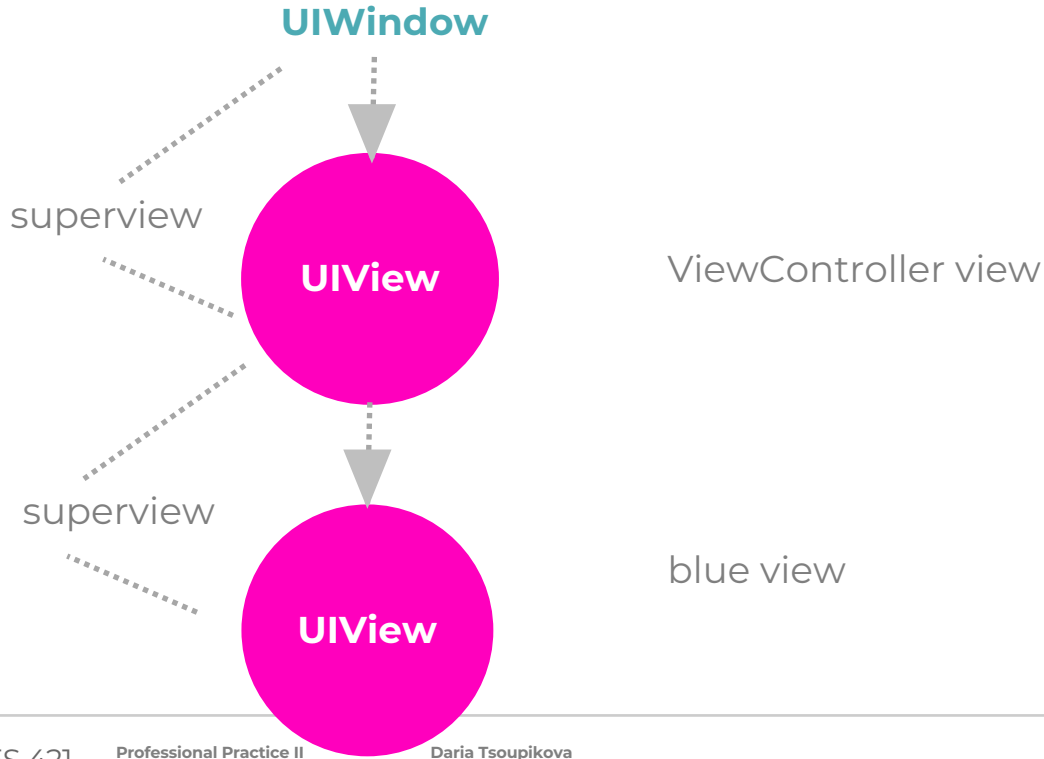
ViewController.swift

WorldTrotter 〉 WorldTrotter 〉 ViewController.swift 〉 viewDidLoad()

```swift
//
//  ViewController.swift
//  WorldTrotter
//
//  Created by daria tsoupikova on 1/11/19.
//  Copyright © 2019 daria tsoupikova. All rights reserved.
//

import UIKit


class ViewController: UIViewController {

    override func viewDidLoad()
    {
        super.viewDidLoad()

        let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
        let firstView = UIView(frame: firstFrame)
        firstView.backgroundColor = UIColor.blue
        view.addSubview(firstView)
    }
}
```

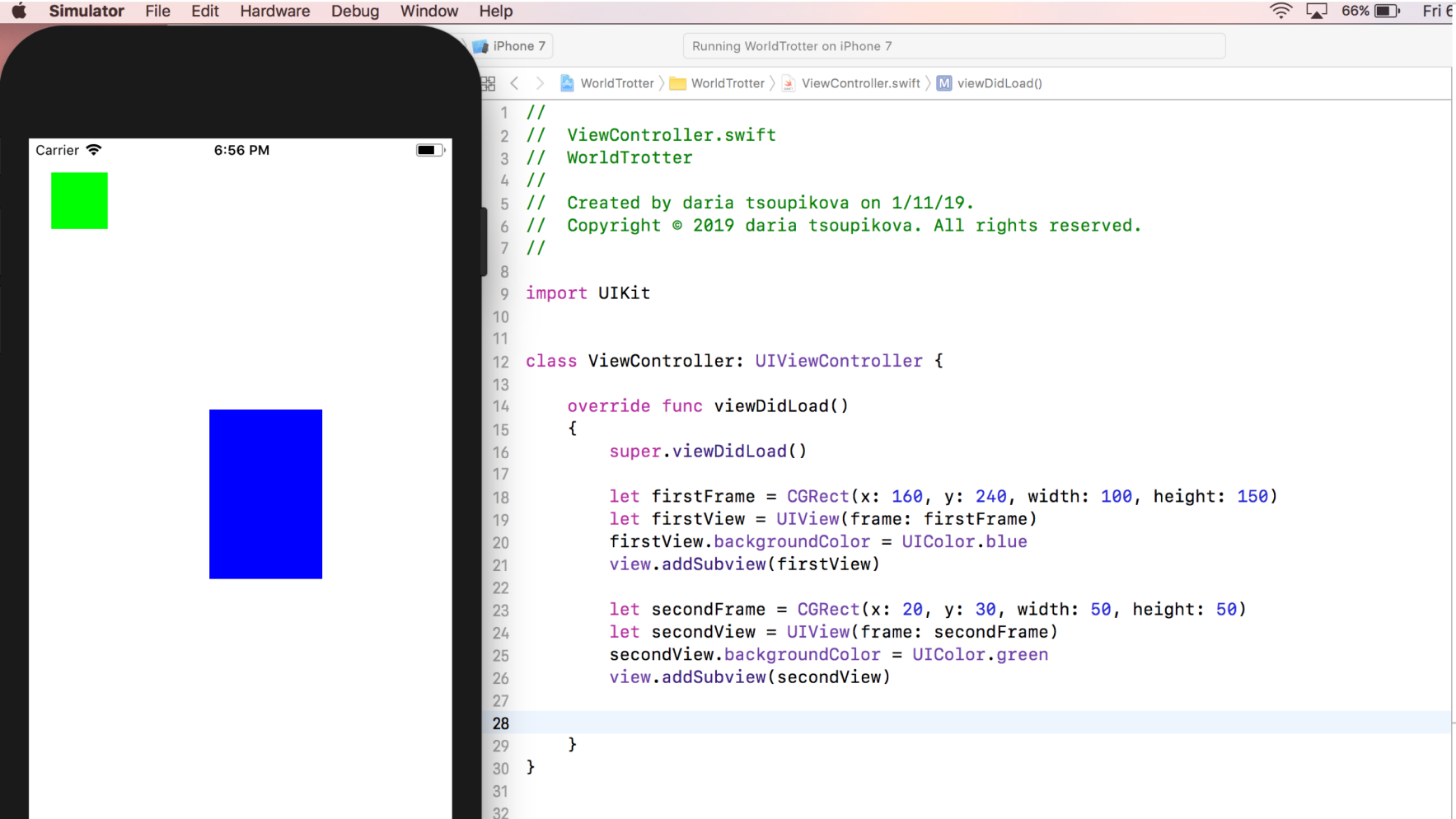
WorldTrotter

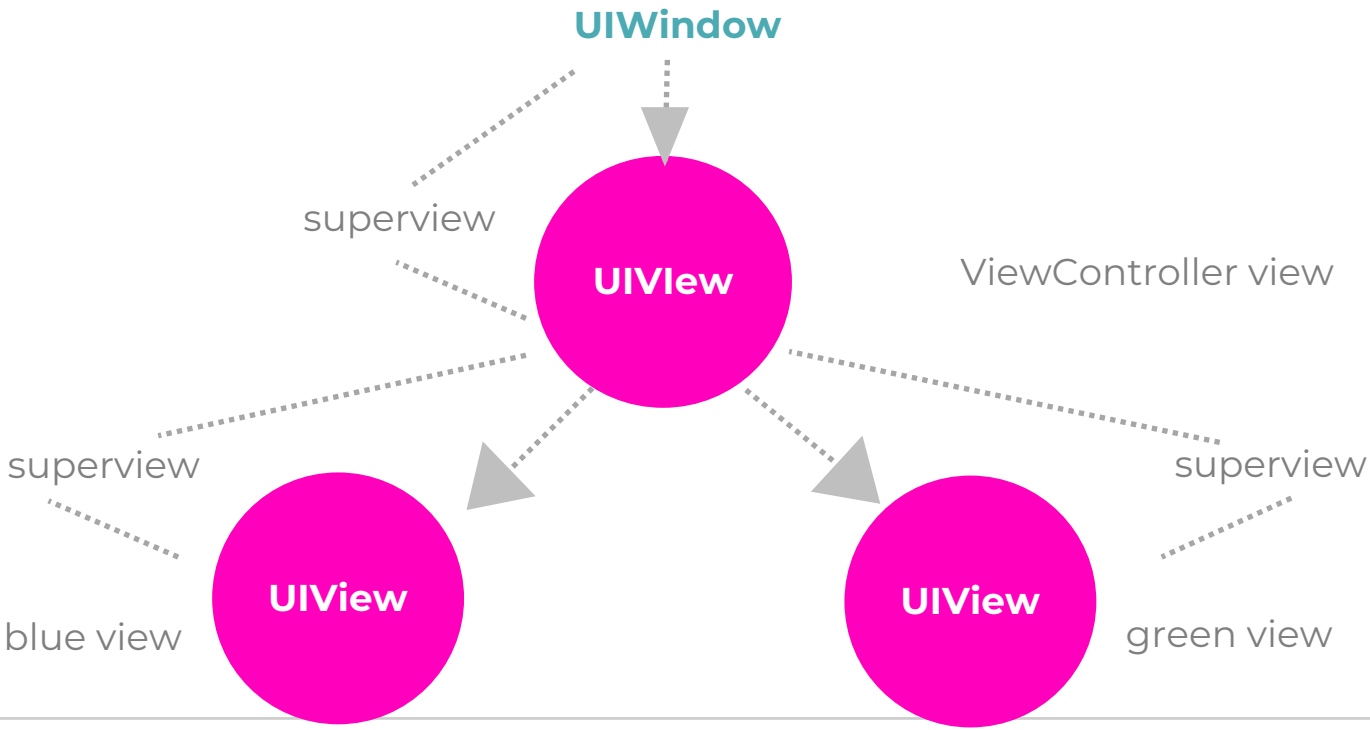
Values are given in points (not pixels!)
If values would be given in px they would not be consistent across the different resolutions.
Any instance of UIView
has a superview property.



Add another instance of UIView:

```
class ViewController: UIViewController {

    override func viewDidLoad()
    {
        super.viewDidLoad()

        let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
        let firstView = UIView(frame: firstFrame)
        firstView.backgroundColor = UIColor.blue
        view.addSubview(firstView)

        let secondFrame = CGRect(x: 20, y: 30, width: 50, height: 50)
        let secondView = UIView(frame: secondFrame)
        secondView.backgroundColor = UIColor.green
        view.addSubview(secondView)

    }
```


Simulator File Edit Hardware Debug Window Help
66%
Fri 6
iPhone 7
Running WorldTrotter on iPhone 7
WorldTrotter 〉 WorldTrotter 〉 ViewController.swift 〉 viewDidLoad()
Carrier
6:56 PM
//
// ViewController.swift
// WorldTrotter
//
// Created by daria tsoupikova on 1/11/19.
// Copyright © 2019 daria tsoupikova. All rights reserved.
//
import UIKit
class ViewController: UIViewController {
override func viewDidLoad()
{
super.viewDidLoad()
let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
let firstView = UIView(frame: firstFrame)
firstView.backgroundColor = UIColor.blue
view.addSubview(firstView)
let secondFrame = CGRect(x: 20, y: 30, width: 50, height: 50)
let secondView = UIView(frame: secondFrame)
secondView.backgroundColor = UIColor.green
view.addSubview(secondView)
}
}

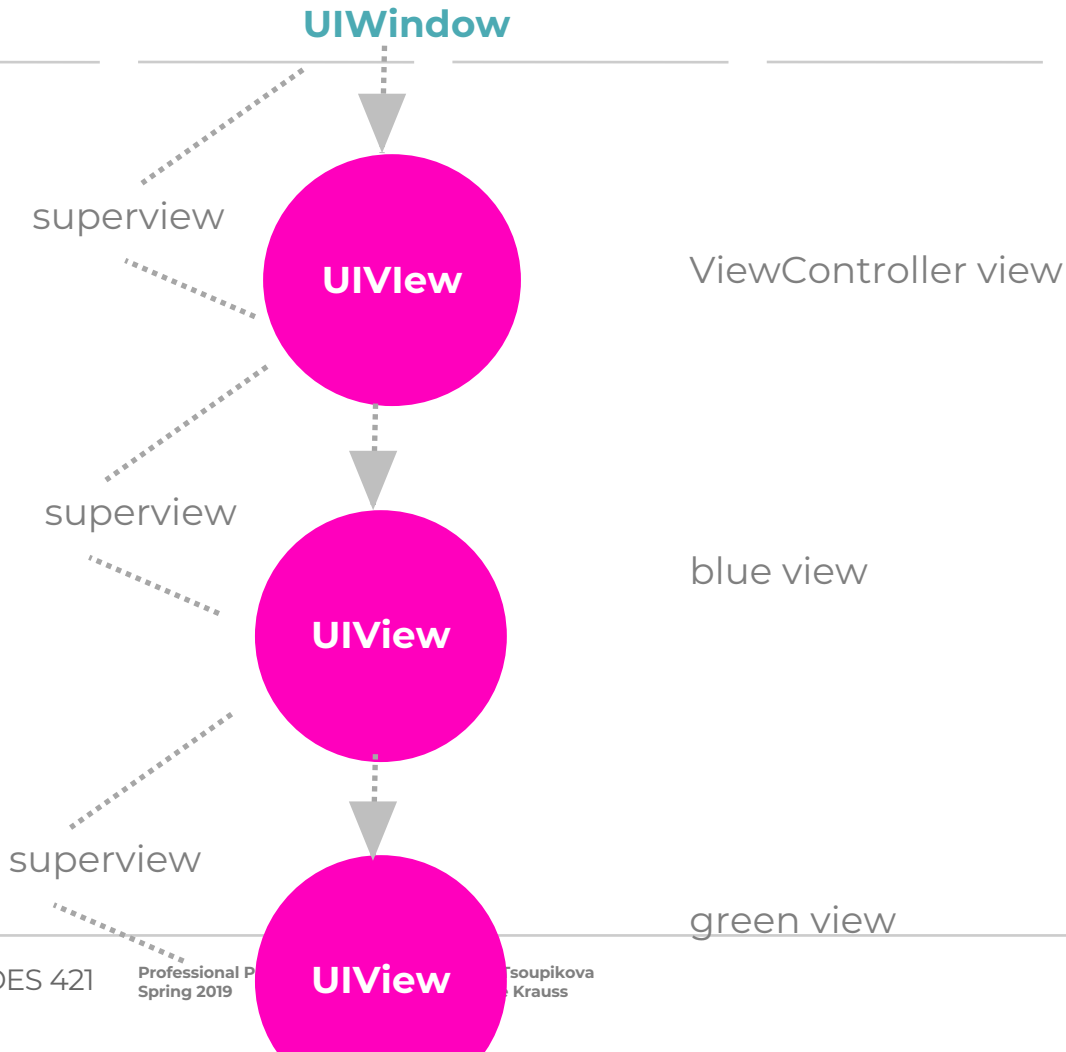

UIWindow
superview
UIView
ViewController view
superview
superview
UIView
UIView
blue view
green view

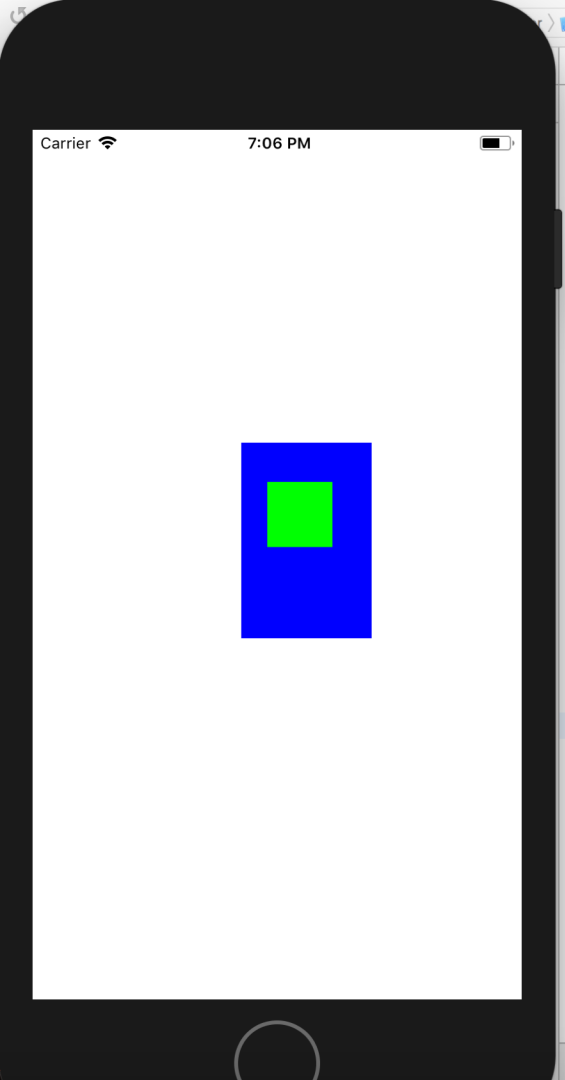
Add another instance of UIView:

```
class ViewController: UIViewController {

    override func viewDidLoad()
    {
        super.viewDidLoad()

        let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
        let firstView = UIView(frame: firstFrame)
        firstView.backgroundColor = UIColor.blue
        view.addSubview(firstView)

        let secondFrame = CGRect(x: 20, y: 30, width: 50, height: 50)
        let secondView = UIView(frame: secondFrame)
        secondView.backgroundColor = UIColor.green
        //view.addSubview(secondView)
        firstView.addSubview(secondView)
```



```
}
```


UIWindow
superview
UIView
ViewController view
superview
UIView
blue view
superview
UIView
green view



iPhone 7 | Running WorldTrotter on iPhone 7

WorldTrotter > WorldTrotter > ViewController.swift > viewDidLoad()

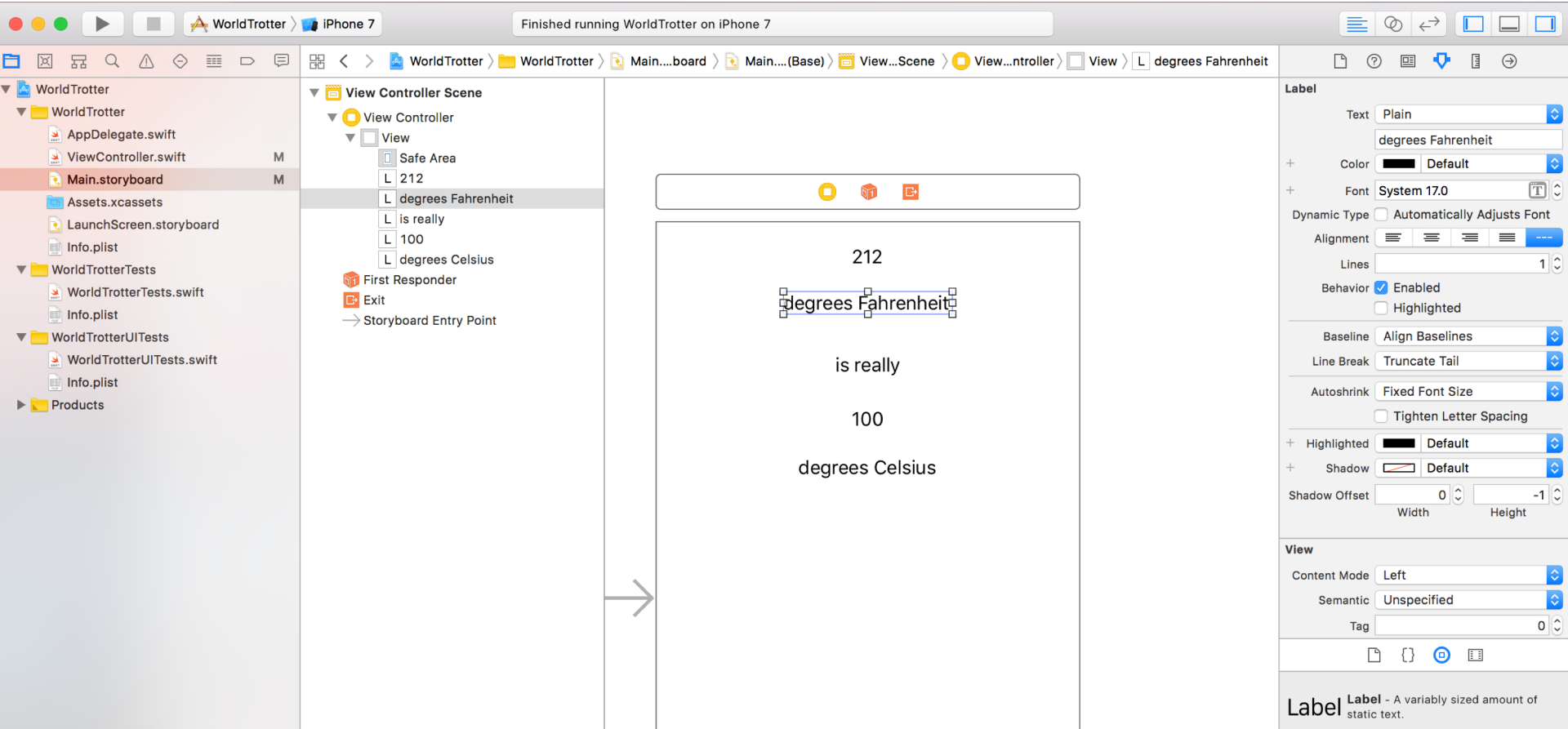
```swift
//
//  ViewController.swift
//  WorldTrotter
//
//  Created by daria tsoupikova on 1/11/19.
//  Copyright © 2019 daria tsoupikova. All rights reserved.
//

import UIKit


class ViewController: UIViewController {

    override func viewDidLoad()
    {
        super.viewDidLoad()

        let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
        let firstView = UIView(frame: firstFrame)
        firstView.backgroundColor = UIColor.blue
        view.addSubview(firstView)

        let secondFrame = CGRect(x: 20, y: 30, width: 50, height: 50)
        let secondView = UIView(frame: secondFrame)
        secondView.backgroundColor = UIColor.green
        //view.addSubview(secondView)
        firstView.addSubview(secondView)



    }
}
```

WorldTrotter

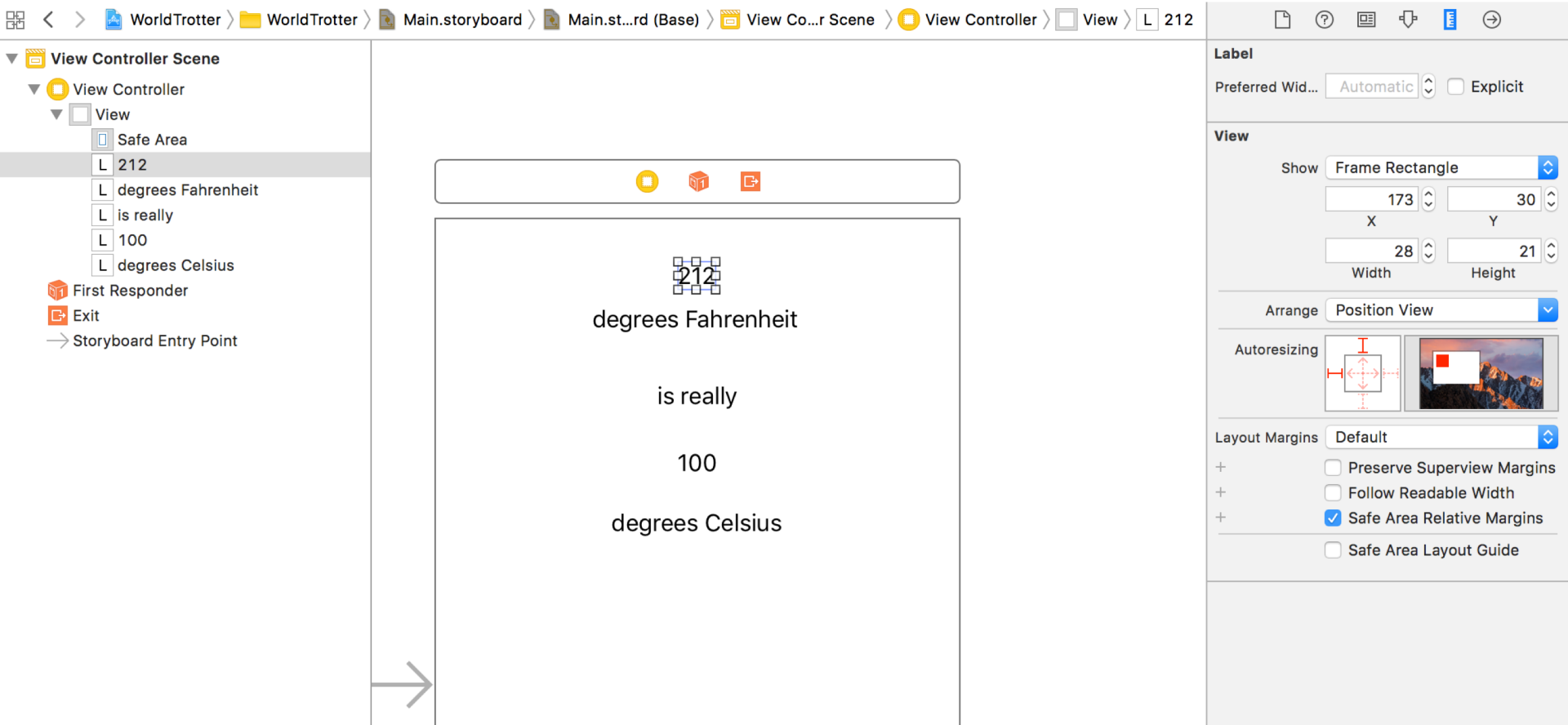
View's frame is relative to its superview, so the top-left corner of secondView is now inset (20, 30) from the top –left corner of the firstView.
Start Building the Interface of the WorldTrotter – remove the code with rectangles

```
class ViewController: UIViewController {

    override func viewDidLoad()
    {
        super.viewDidLoad()

        let firstFrame = CGRect(x: 160, y: 240, width: 100, height: 150)
        let firstView = UIView(frame: firstFrame)
        firstView.backgroundColor = UIColor.blue
        view.addSubview(firstView)

        let secondFrame = CGRect(x: 20, y: 30, width: 50, height: 50)
        let secondView = UIView(frame: secondFrame)
        secondView.backgroundColor = UIColor.green
        //view.addSubview(secondView)
        firstView.addSubview(secondView)

    }
}
```

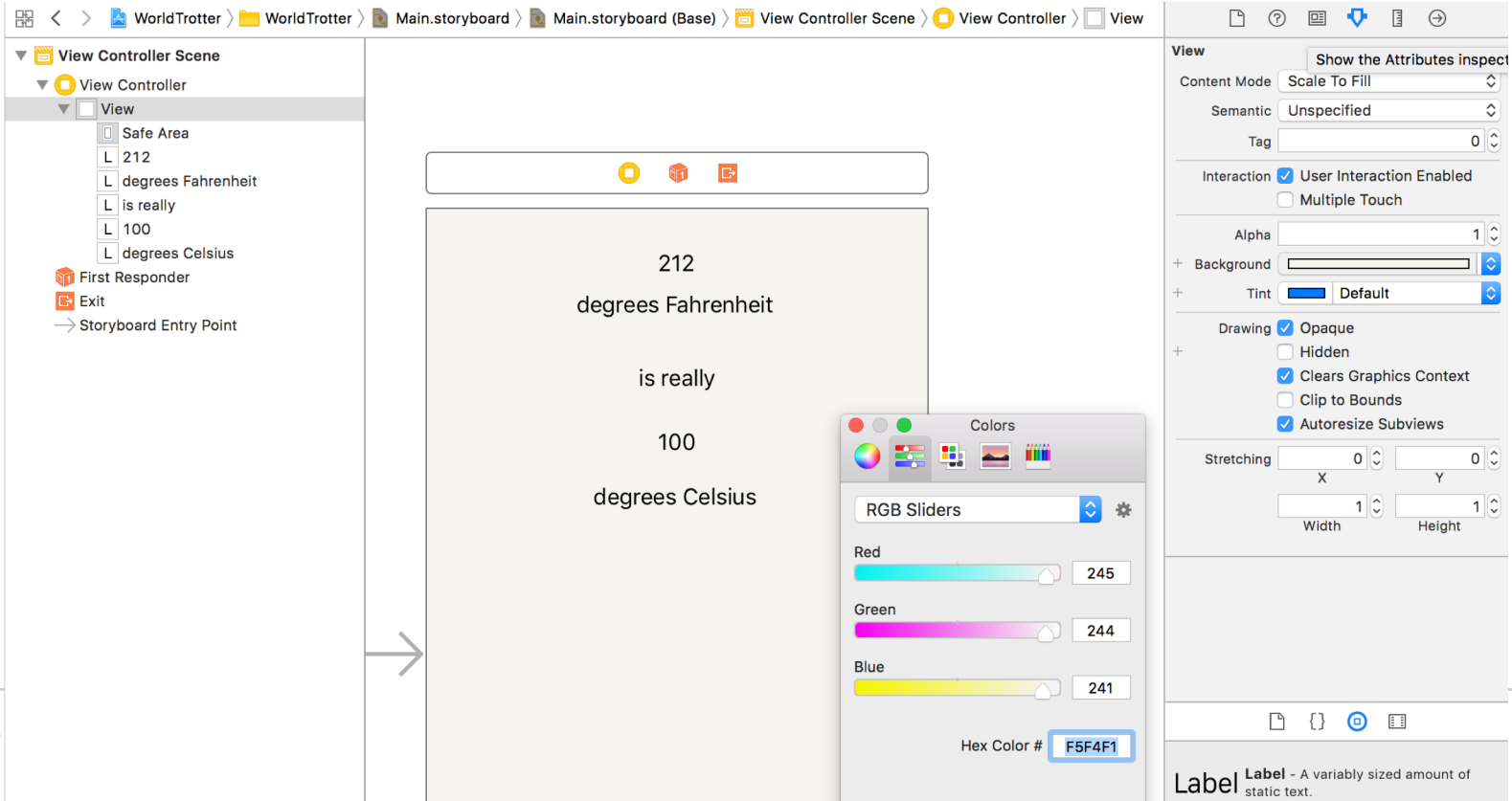
Open Main. Storyboard. Create 5 UILabels, center them horizontally on top.



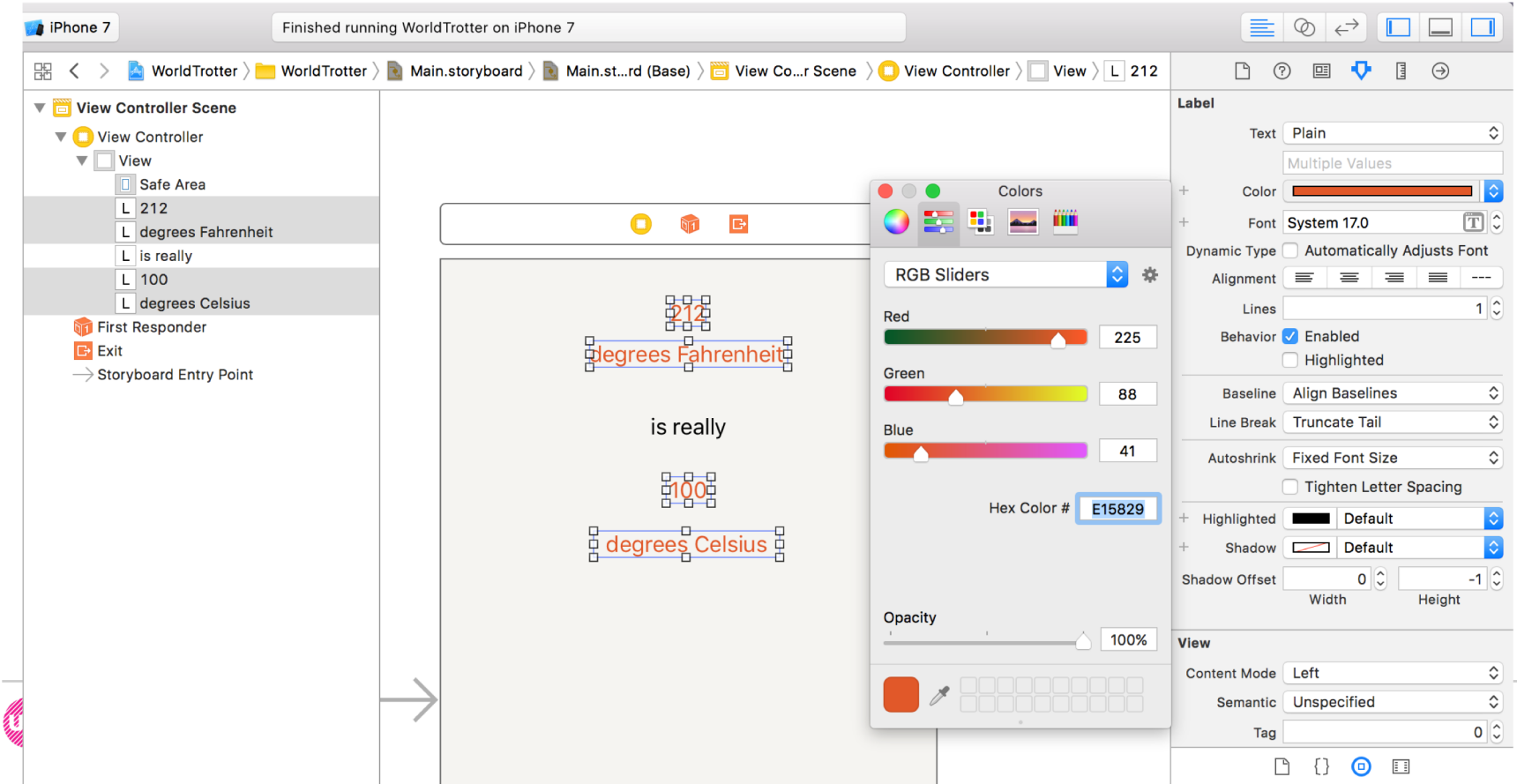
View Frame values, test on simulator- should look identical.



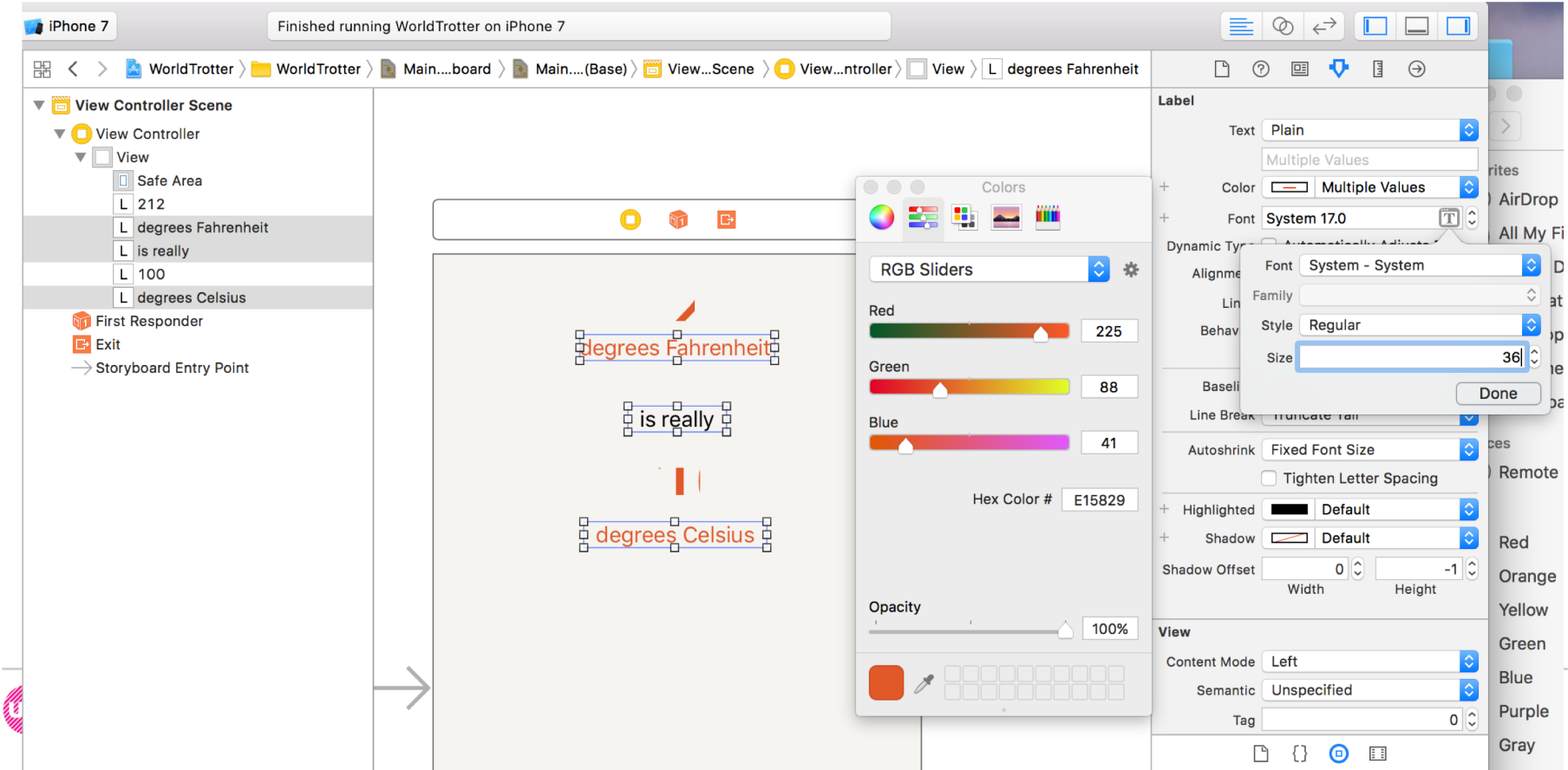
Customize the view properties. Select bg in the storyboard. Attributes Inspector> Background color> RGB sliders > Hex F5F4F1



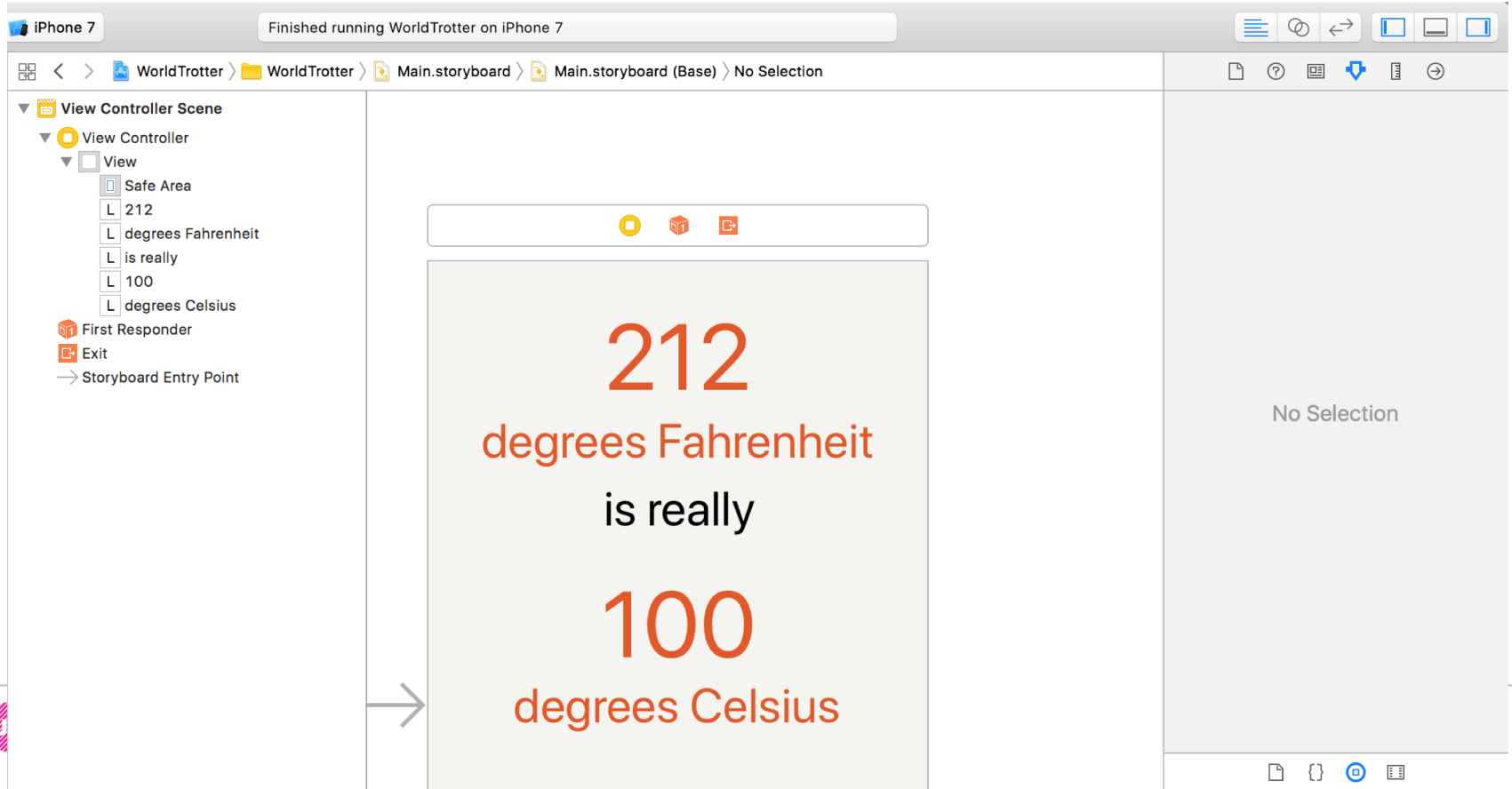
Select top two and bottom two labels >Color>E15829



212 and 100 labels > Font Size > 70. Degrees F and Degrees C > Font>Size > 36



Select all labels > Scale to fit using (Command -=) Arrange vertically. Test in simulator



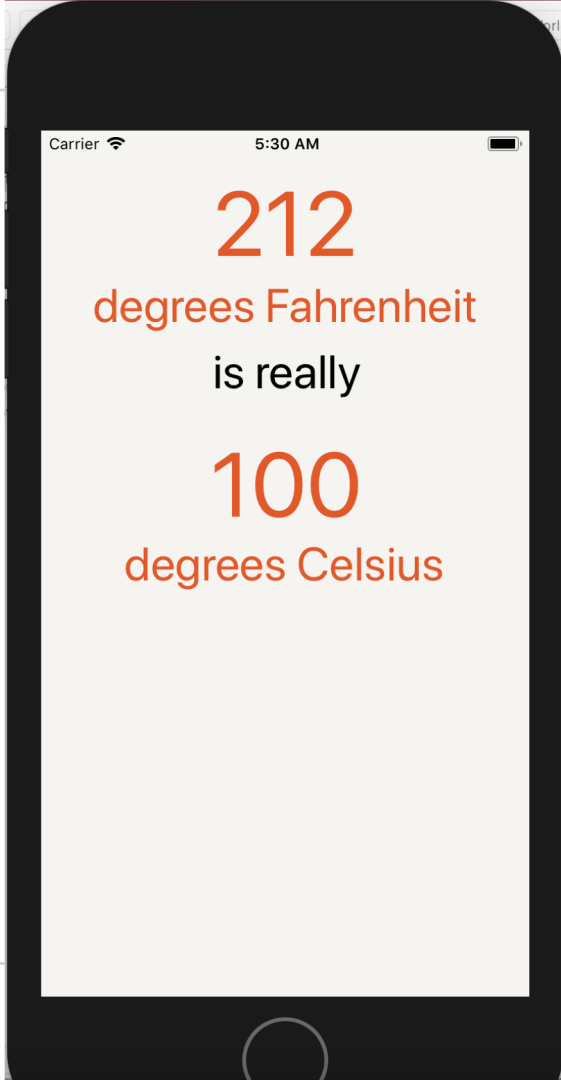
The labels will appear slightly shifted to the left.

Absolute Frames have major problems:

- When content is resized, the frames do no update automatically.
- The view does not look equally good on different screen sizes

Do not use absolute frame for views.
Use Auto Layout which flexibly computes frames based on the constraints specified for each view.

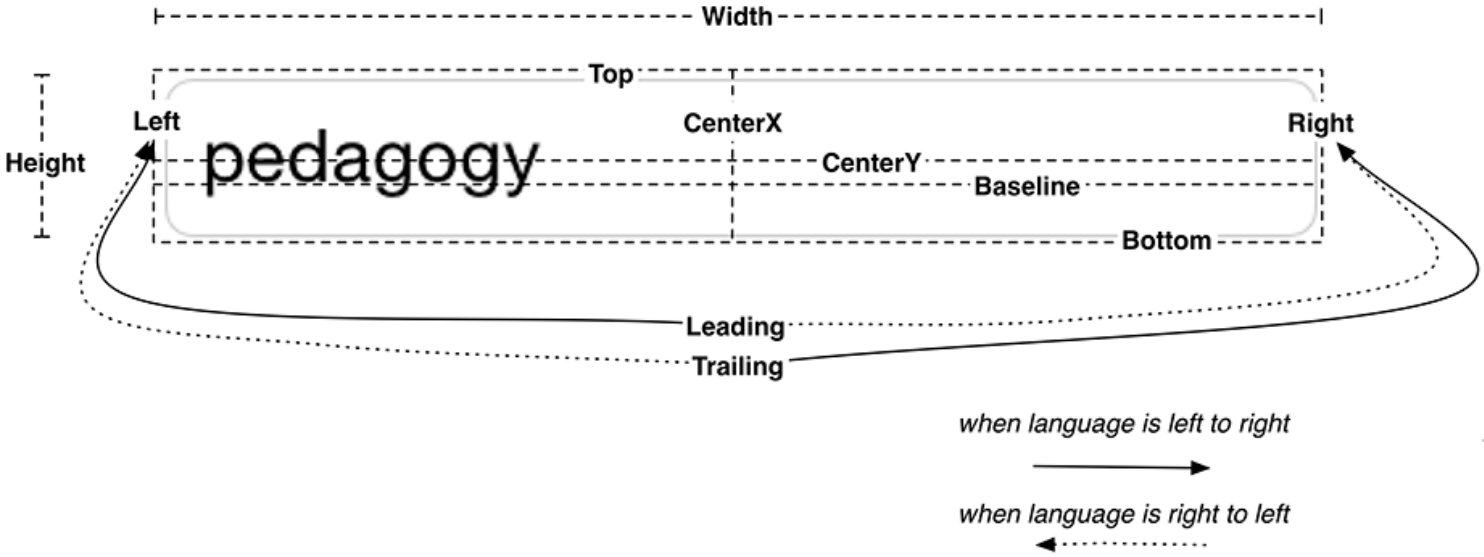
Label should remain the same distance from the top of the screen; and horizontally centered within their superview. They should update If font or text or labels change.



Auto Layout describes the layout of your views in a relative way that enables their frames to be determined at a runtime, so that the frames' definitions can take into account the screen of the device that the app is running on.

Auto Layout allows Responsive design

The Auto Layout system is based on the *alignment rectangle*. This rectangle is defined by several *layout attributes*



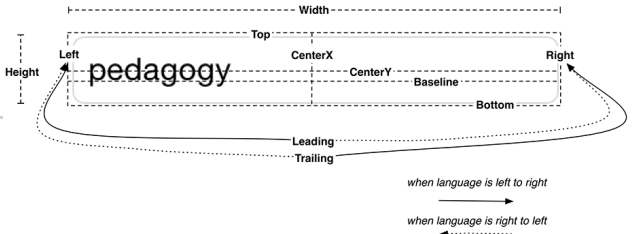
Width / height determine the alignment rectangle's size.

Top/Bottom/Left/Right determine the spacing between the given edge of the alignment rectangle and the alignment rectangle of another view in the hierarchy.

CenterX/CenterY Determine the center point of the alignment rectangle.

Baseline This value is the same as the bottom attribute for most, but not all, views. For example, **UITextField** defines its baseline as the bottom of the text it displays rather than the bottom of the alignment rectangle. This keeps "descenders" (letters like 'g' and 'p' that descend below the baseline) from being obscured by a view right below the text field. e the center point of the alignment rectangle.

Leading/Trailing These values are language-specific attributes. If the device is set to a language that reads left to right (e.g., English), then the leading attribute is the same as the left attribute and the trailing attribute is the same as the right attribute. If the language reads right to left (e.g., Arabic), then the leading attribute is on the right and the trailing attribute is on the left. **Interface Builder**automatically prefers leading and trailing over left and right, and, in general, you should as well.



By default, every view has an alignment rectangle, and every view hierarchy uses Auto Layout.

The alignment rectangle is very similar to the frame. In fact, these two rectangles are often the same. Whereas the frame encompasses the entire view, the alignment rectangle only encompasses the content that you wish to use for alignment purposes. Frame vs. alignment rectangle:

Frame

Alignment rectangle

You cannot define a view's alignment rectangle directly. You do not have enough information (like screen size) to do that. Instead, you provide a set of *constraints*. Taken together, these constraints enable the system to determine the layout attributes, and thus the alignment rectangle, for each view in the view hierarchy.

A *constraint* defines a specific relationship in a view hierarchy that can be used to determine a layout attribute for one or more views. For example, you might add a constraint like, "The vertical space between these two views should always be 8 points," or, "These views must always have the same width." A constraint can also be used to give a view a fixed size, like, "This view's height should always be 44 points."

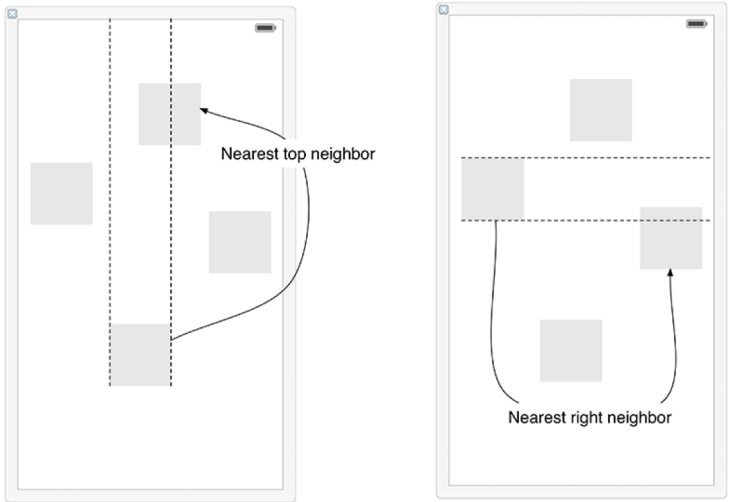
You do not need a constraint for every layout attribute. Some values may come directly from a constraint; others will be computed by the values of related layout attributes. For example, if a view's constraints set its left edge and its width, then the right edge is already determined (left edge + width = right edge, always). As a general rule of thumb, you need at least two constraints per dimension (horizontal and vertical).

If, after all of the constraints have been considered, there is still an ambiguous or missing value for a layout attribute, then there will be errors and warnings from Auto Layout and your interface will not look as you expect on all devices. Debugging these problems is important/

First, describe what you want the view to look like independent of screen size. For example, you might say that you want the top label to be:

- 8 points from the top of the screen
- centered horizontally in its superview
- as wide and as tall as its text

To turn this description into constraints in **Interface Builder**, it will help to understand how to find a view's *nearest neighbor*. The nearest neighbor is the closest sibling view in the specified direction.
If a view does not have any siblings in the specified direction, then the nearest neighbor is its superview, also known as its container.



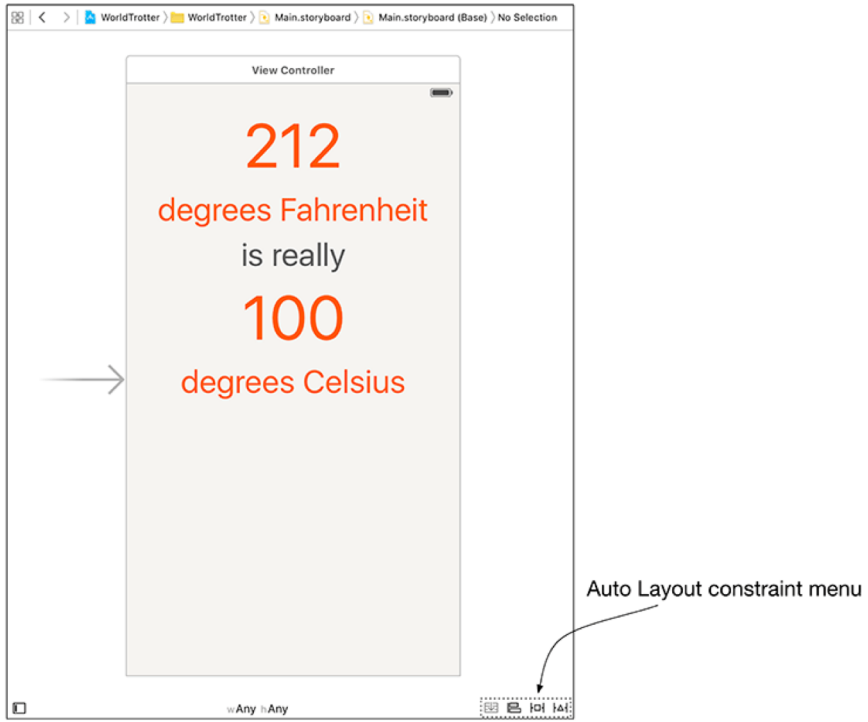
the constraints for the label:

- The label's top edge should be 8 points away from its nearest neighbor (which is its container – the view of the **ViewController**).
- The label's center should be the same as its superview's center.
- The label's width should be equal to the width of its text rendered at its font size.
- The label's height should be equal to the height of its text rendered at its font size.

If you consider the first and fourth constraints, you can see that there is no need to explicitly constrain the label's bottom edge. It will be determined from the constraints on the label's top edge and the label's height. Similarly, the second and third constraints together determine the label's right and left edges.

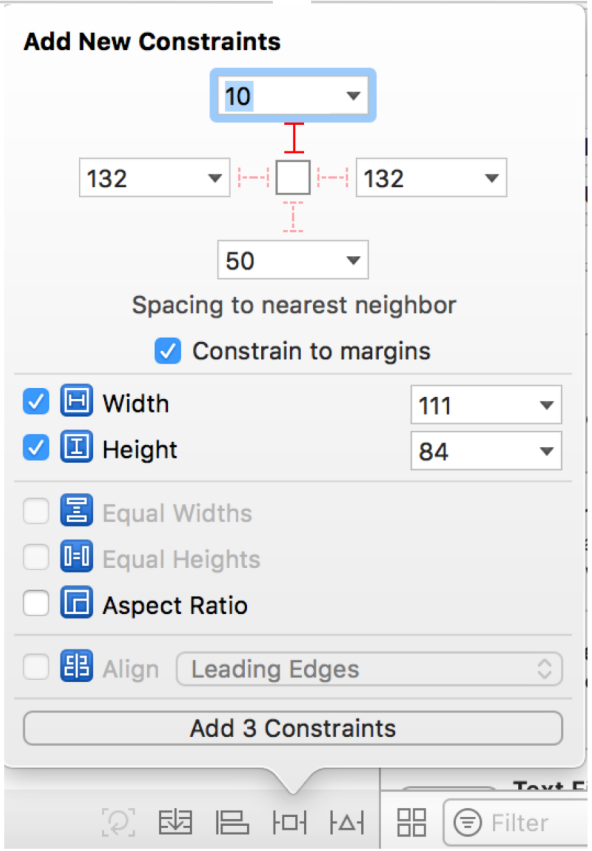
Constraints can be added using **Interface Builder** or in code. Apple recommends that you add constraints using **Interface Builder** whenever possible, and that is what you will do here. However, if your views are created and configured programmatically, then you can add constraints in code.

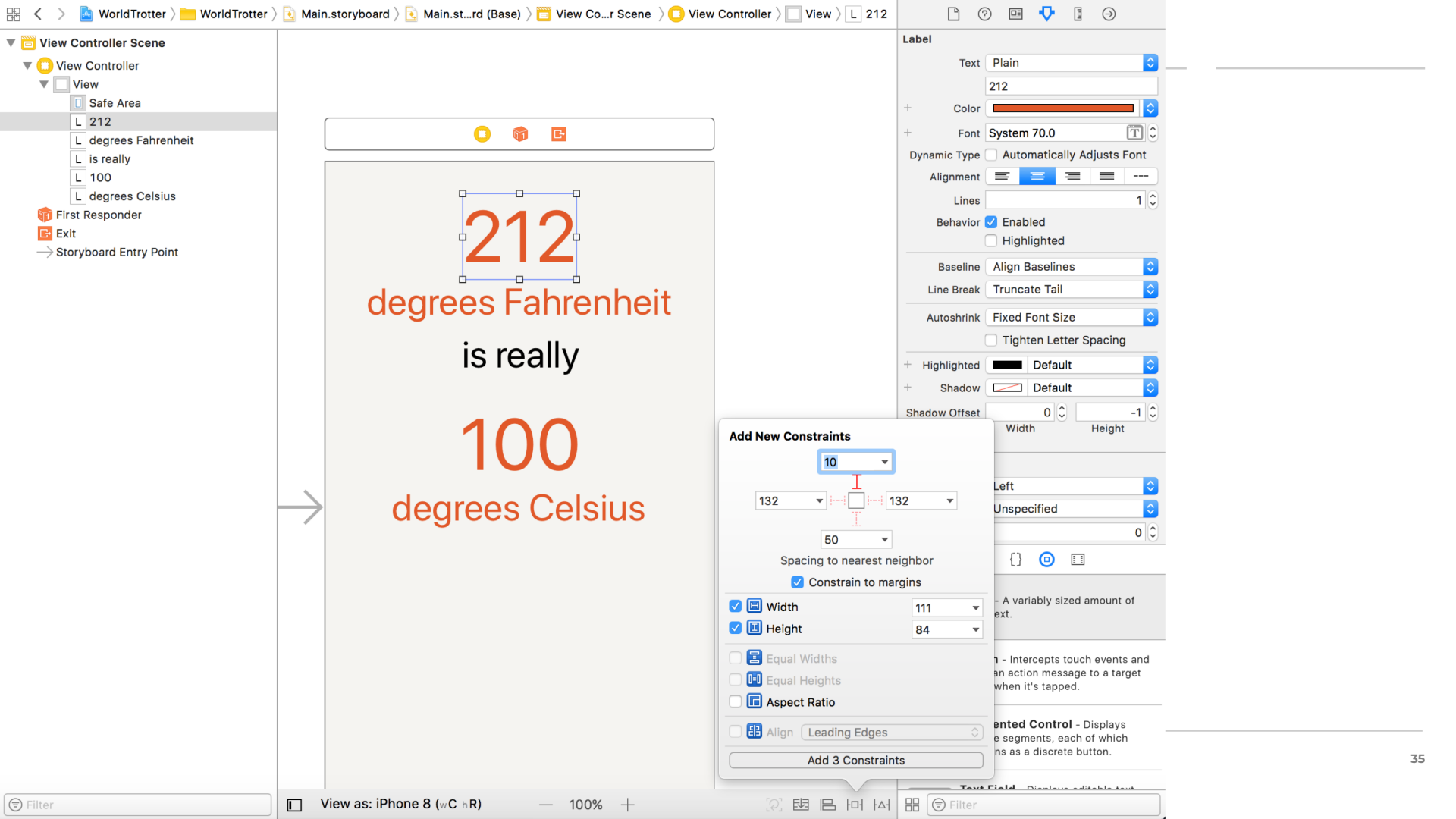
Select the top label on the canvas. In the bottom righthand corner of the canvas, find the Auto Layout constraint menu 



At the top of the **Pin** menu are four values that describe the label's current spacing from its nearest neighbor on the canvas. For this label, you are only interested in the top value.
To turn this value into a constraint, click the top red strut separating the value from the square in the middle. The strut will become a solid red line.

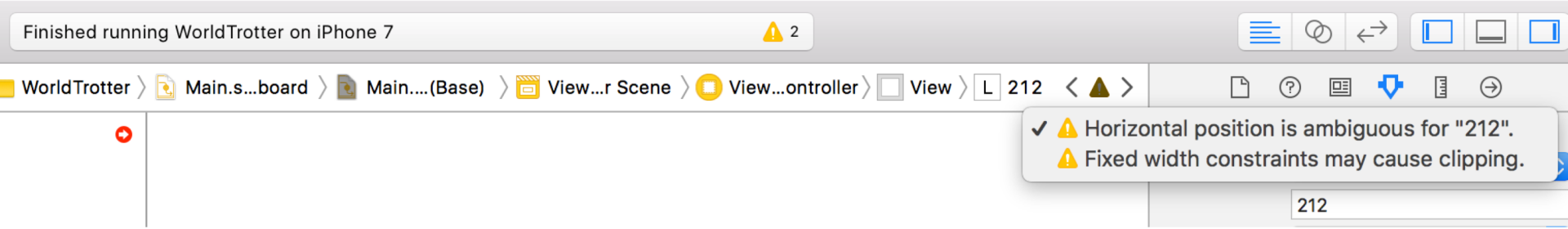
In the middle of the menu, find the label's **Width** and **Height**. The values next to **Width** and **Height** indicate the current canvas values. To constrain the label's width and height to the current canvas values, check the boxes next to **Width** and **Height**.
Click button **Add 3 Constraints**.




WorldTrotter
WorldTrotter
Main.storyboard
Main.st...rd (Base)
View Co...r Scene
View Controller
View
212
View Controller Scene
View Controller
View
Safe Area
212
degrees Fahrenheit
is really
100
degrees Celsius
First Responder
Exit
Storyboard Entry Point
212
degrees Fahrenheit
is really
100
degrees Celsius
Label
Text
Plain
212
Color
Font
System 70.0
Dynamic Type
Automatically Adjusts Font
Alignment
Lines
1
Behavior
Enabled
Highlighted
Baseline
Align Baselines
Line Break
Truncate Tail
Autoshrink
Fixed Font Size
Tighten Letter Spacing
Highlighted
Default
Shadow
Default
Shadow Offset
0
-1
Width
Height
Add New Constraints
10
132
132
50
Spacing to nearest neighbor
Constrain to margins
Width
111
Height
84
Equal Widths
Equal Heights
Aspect Ratio
Align
Leading Edges
Add 3 Constraints
Left
Unspecified
0
- A variably sized amount of
ext.
- Intercepts touch events and
an action message to a target
when it's tapped.
ented Control - Displays
e segments, each of which
ns as a discrete button.
Filter
View as: iPhone 8 (wC hR)
100%
Filter

At this point, you have not specified enough constraints to fully determine the alignment rectangle. **Interface Builder** will help you determine what the problem is.

In the top right corner of **Interface Builder**, notice the yellow warning sign. Click on this icon to reveal the issue: “Horizontal position is ambiguous for "212".”

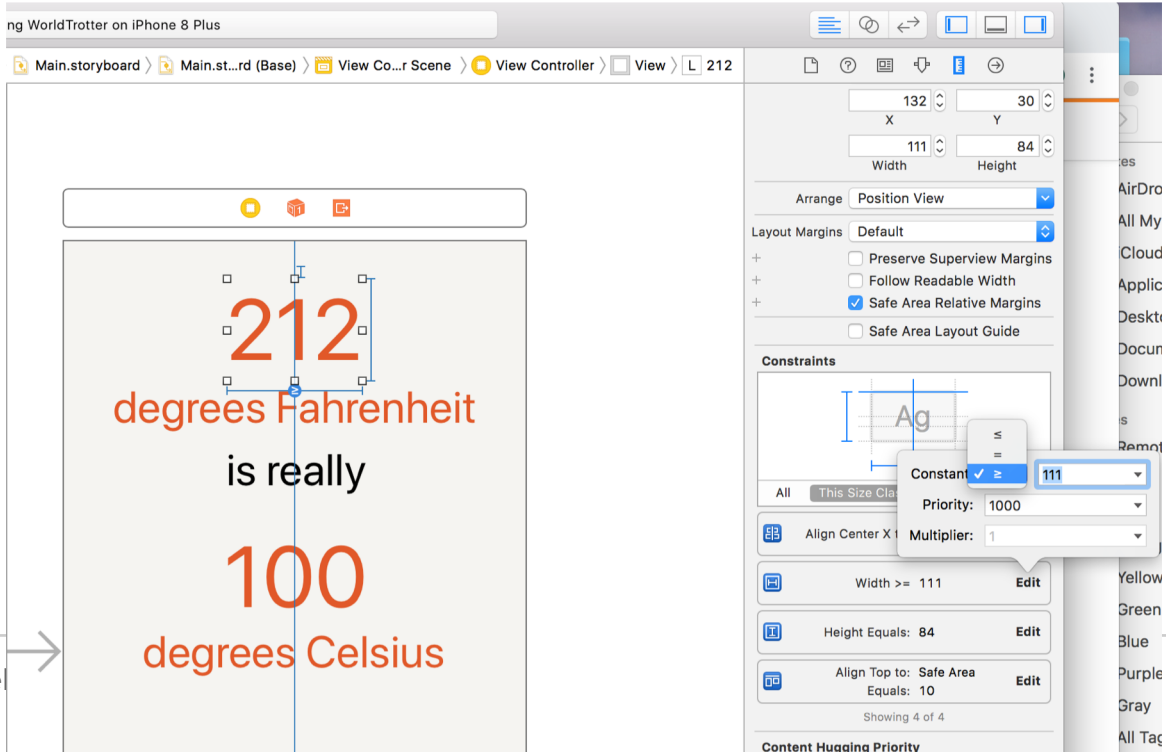


With the top label still selected, click the align_.  icon (the second from the left in the Auto Layout constraints menu) to reveal the Align menu. If you have multiple views selected, this menu will allow you to align attributes among the views. Since you have only selected one label, the only options you are given are to align the view within its container.

select Horizontally in Container (do not click Add 1 Constraint yet). Once you add this constraint, there will be enough constraints to fully determine the alignment rectangle. To ensure that the frame of the label matches the constraints specified, open the Update Frames pop-up menu from the Align menu and select Items of New Constraints. This will reposition the label to match the constraints that have been added. Now click on Add 1 Constraint to add the centering constraint and reposition the label.

The label's constraints are all blue now that the alignment rectangle for the label is fully specified. Additionally, the warning at the top right corner of Interface Builder is now gone.

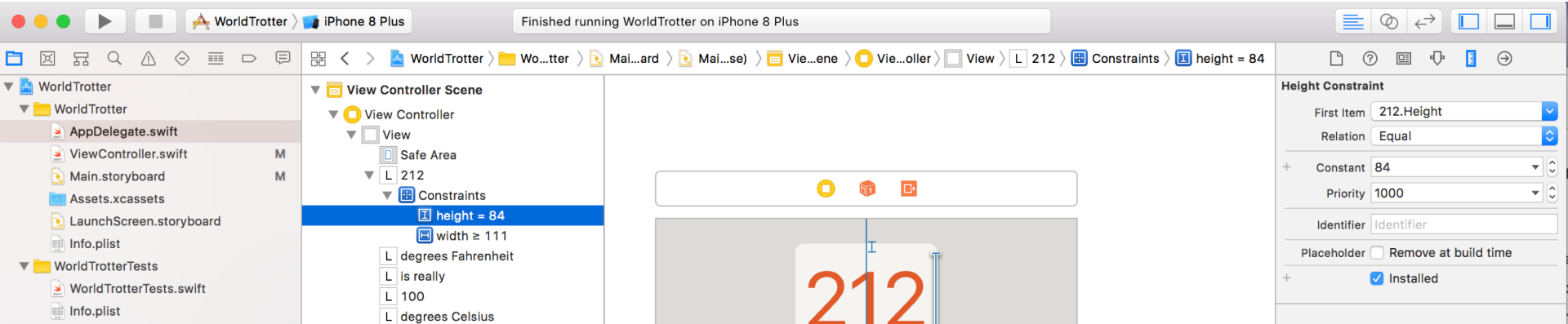
To fix the fixed-width warnings in your app - change the width of the object spacings from fixed width to greater than or equal or less than or equal. (<=. Or >=) select the object in interface builder > go to the size inspector > constraints > width>changing to <=



Because we added explicit width and height constraints to the label, its size is not flexible.
If the text or font were to change, the label will not look centered.

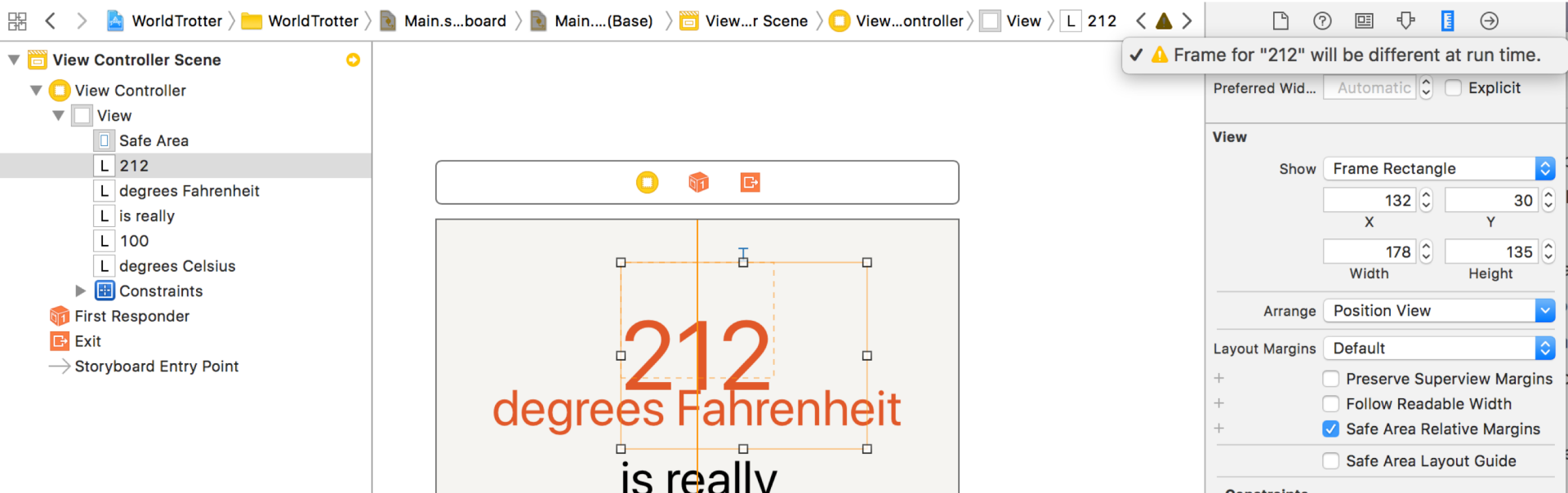
The intrinsic content size is the size that a view “wants” to naturally be.
For labels, this size is the size of the text rendered at the given font. For images, this is the size of the image itself. A view’s intrinsic content size acts as implicit width and height constraints.
If you do not specify constraints that explicitly determine the width, the view will be its intrinsic width. The same goes for the height.

To remove the explicit width and height constraints:
select the width constraint on the label, press the Delete key. Do the same for the height constraint.



Blue constraints indicate that the alignment rectangle for a view is fully specified. Orange constraints often indicate a misplaced view. This means that the frame for the view in Interface Builder is different than the frame that Auto Layout has computed.

Resize the top label on the canvas using the resize controls and look for the yellow warning in the top right corner of the canvas. Click on this warning icon to reveal the problem: “Frame for "212" will be different at run time”



As the warning says, the frame at runtime will not be the same as the frame specified on the canvas.
an orange dotted line that indicates what the runtime frame will be.

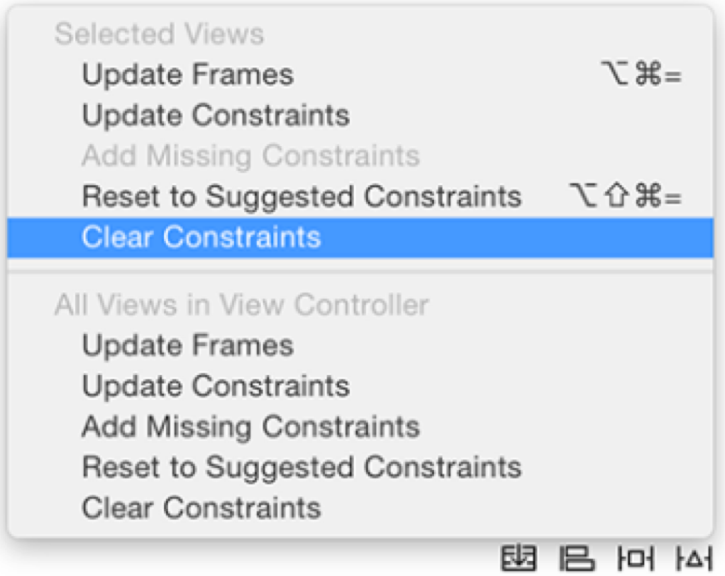
Build and run the application. Notice that the label is still centered despite the new frame that you gave it in Interface Builder. However the disconnect between what you have specified in Interface Builder and the constraints computed by Auto Layout will cause problems down the line as you continue to build your views.

Ro fix the misplaced view:

- select the top label on the canvas
- click Update Frames icon. This will update the frame of the label to match the frame that the constraints will compute.

Word of caution: if you try to update the frames for a view that does not have enough constraints, you will almost certainly get unexpected results. If that happens, undo the change and inspect the constraints to see what is missing.

Select the top label on the canvas. Open the Resolve Auto Layout Issues menu and select Clear Constraints from the Selected Views section.



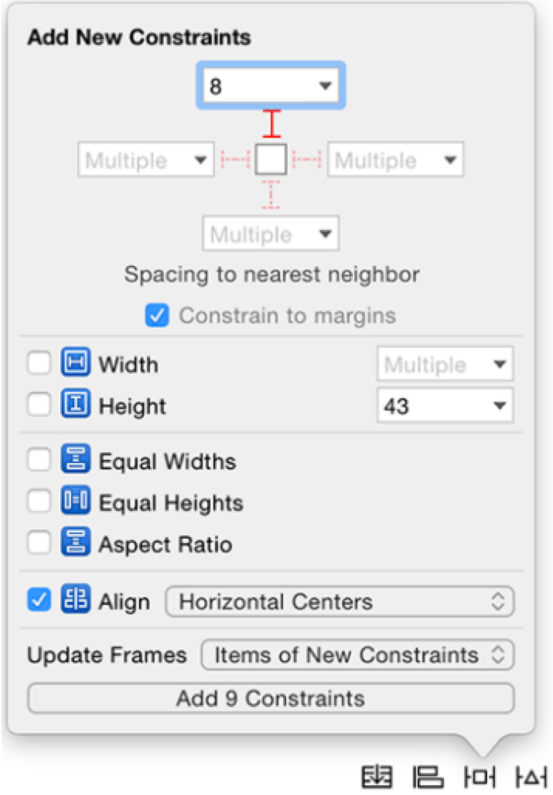
to add the constraints to all of the views in two steps:

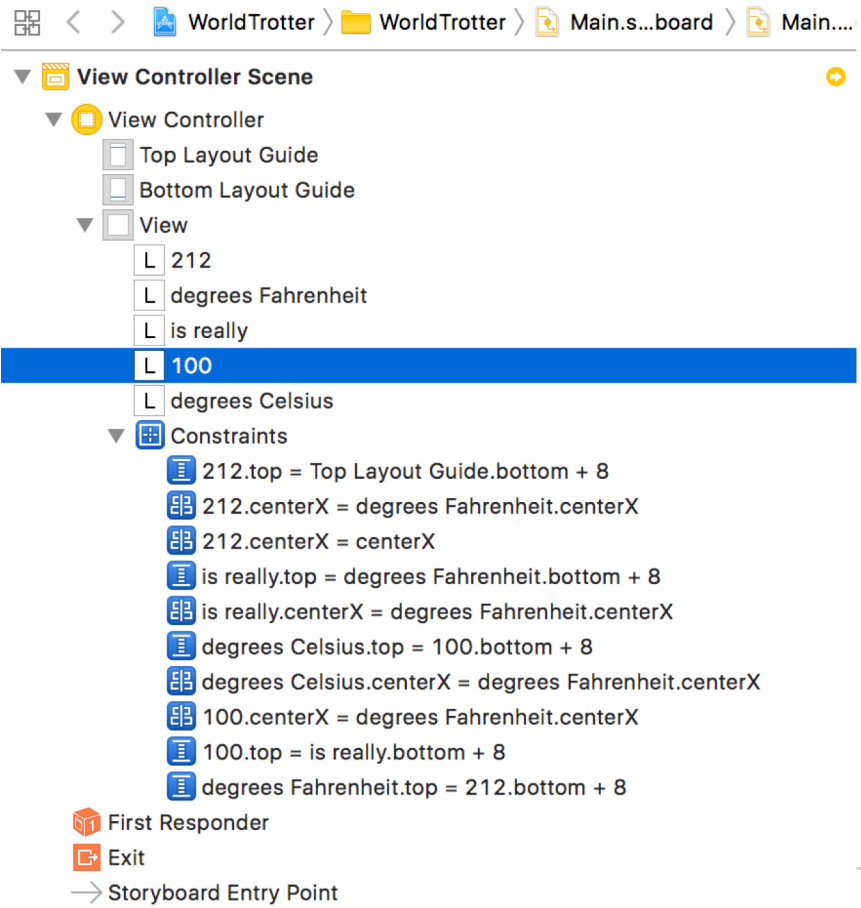
1. center the top label horizontally within the superview.
2. add constraints that pin the top of each label to its nearest neighbor while aligning the centers of all of the labels.

Select the top label. Open the Align menu and choose Horizontally in Container with a constant of 0. *Make sure that Update Frames has None selected; remember that you do not want to update the frame of a view that does not have enough constraints, and this one constraint will certainly not provide enough information to compute the alignment rectangle.*
Add 1 Constraint.

select all five labels on the canvas to add constraints to multiple views simultaneously.
Open Add Constraints menu :

- Select the top strut and make sure it has a constant of 8.
- From the Align menu, choose Horizontal Centers.
- From the Update Frames menu, choose Items of New Constraints.




WorldTrotter
WorldTrotter
Main.s...board
Main....
View Controller Scene
View Controller
Top Layout Guide
Bottom Layout Guide
View
212
degrees Fahrenheit
is really
100
degrees Celsius
Constraints
212.top = Top Layout Guide.bottom + 8
212.centerX = degrees Fahrenheit.centerX
212.centerX = centerX
is really.top = degrees Fahrenheit.bottom + 8
is really.centerX = degrees Fahrenheit.centerX
degrees Celsius.top = 100.bottom + 8
degrees Celsius.centerX = degrees Fahrenheit.centerX
100.centerX = degrees Fahrenheit.centerX
100.top = is really.bottom + 8
degrees Fahrenheit.top = 212.bottom + 8
First Responder
Exit
Storyboard Entry Point

Assignment 1

Chapter 4 “Text Input and Delegation”
pp. 69-85

Add keyboard functionality to you app

Good Code is as little code as possible.