CS275
Fall 2020

iOS: Programmatic Views

BNRG CHAPTER 6

Creating a View from Code

We have seen an example of how to build the view for a view controller in the storyboard, using Interface Builder

It's also possible to create a view in the code for the app

Creating a View from Code

`loadView()` is called when a view controller is created

At that point, that view controller's `view` property is nil (i.e., it doesn't yet have a view)

property: what other languages call a member variable of a class

Creating a View from Code

- To set the view, set the `view` property of the view controller
- Override the `loadView()` method and set the view there

```
import UIKit
import MapKit

class MapViewController: UIViewController {
    var mapView: MKMapView!

    override func viewDidLoad() {
        super.viewDidLoad()
        print("MapViewController loaded its view.")
    }

    override func loadView() {
        print("hello from loadView()")
        // create a map view
        mapView = MKMapView()
        // set it as *the* view of this view controller
        view = mapView
    }
}
```

Programmatic Constraints

If we create a view in Interface Builder (IB), then we can use Auto Layout to build the constraints for the elements of the view

If instead we create a view in code, then we have to create the constraints in code also

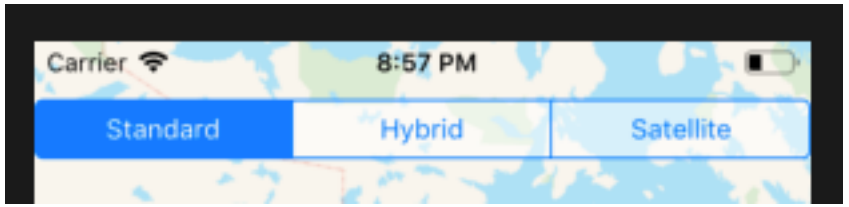
Words of wisdom: "Apple recommends that you care and constrain your views in IB whenever possible"

Segmented Control

A segmented control: "A horizontal control made of multiple segments, each segment functioning as a discrete button"

To add a segmented control (an instance of `UISegmentedControl`), we need to specify its appearance and behavior, from code:

- the labels for the individual items
- the background color
- the option that is initially selected



Segmented Control

Here's the code:

```
let segmentedControl = UISegmentedControl(items: ["Standard", "Hybrid", "Satellite"])
segmentedControl.backgroundColor = UIColor.white.withAlphaComponent(0.5)
segmentedControl.selectedSegmentIndex = 0

segmentedControl.translatesAutoresizingMaskIntoConstraints = false
view.addSubview(segmentedControl)
```

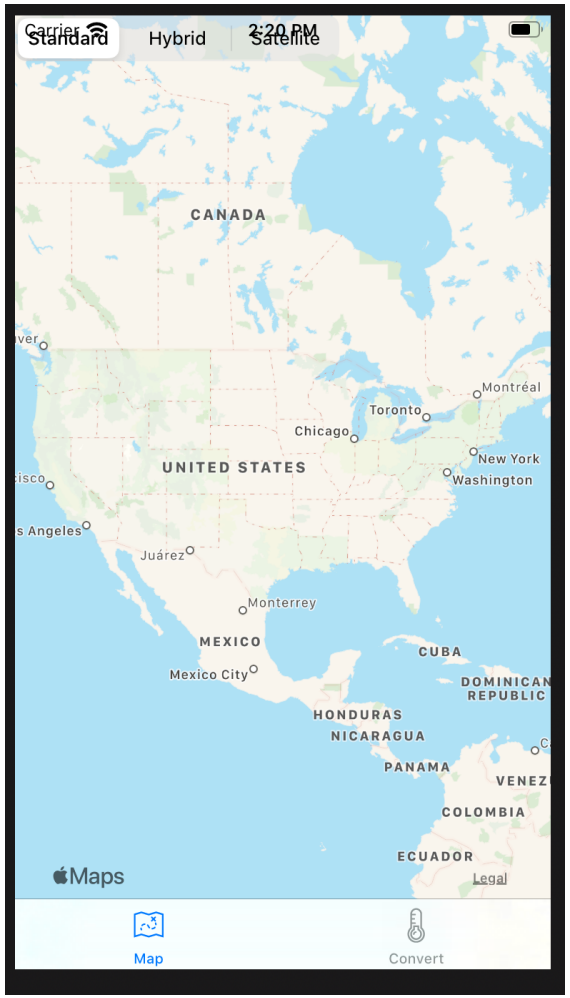
autoresizing masks

- a system for layout creation in iOS from before Auto Layout existed
- turn it off, because with Auto Layout (and static or programmatic constraints), this system does not play well

Segmented Control

Now if we build and run, we get this:

There's something not quite right!



Anchors and Constraints

When we create a view programmatically, we must constrain it

Anchors: properties on a view that correspond to attributes that we might use in order to constrain the view

- for example: leading, top, trailing, bottom

Anchors and Constraints

Example constraints:

- the top anchor of the segmented control should be equal to the top anchor of its superview
- the leading anchor of the segmented control should be equal to the leading anchor of its superview
- the trailing anchor of the segmented control should be equal to the trailing anchor of its superview

Anchors and Constraints

Example constraints:

- the top anchor of the segmented control should be equal to the top anchor of its superview

```
let topConstraint = segmentedControl.topAnchor.constraint(equalTo: view.topAnchor)
```

- the leading anchor of the segmented control should be equal to the leading anchor of its superview

```
let leadingConstraint = segmentedControl.leadingAnchor.constraint(equalTo: view.leadingAnchor)
```

- the trailing anchor of the segmented control should be equal to the trailing anchor of its superview

```
let trailingConstraint = segmentedControl.trailingAnchor.constraint(equalTo: view.trailingAnchor)
```

Anchors and Constraints

Can also use a constant within the constraint:

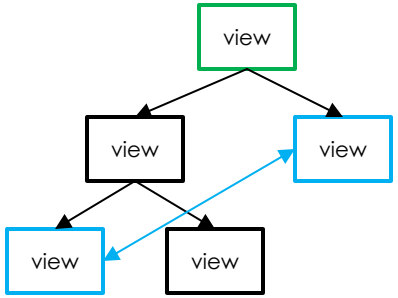
```
let topConstraint = segmentedControl.topAnchor.constraint(equalTo: view.topAnchor, constant: 20)
```

Activating Constraints

Last thing: must activate the constraints:

```
topConstraint.isActive = true
leadingConstraint.isActive = true
trailingConstraint.isActive = true
```

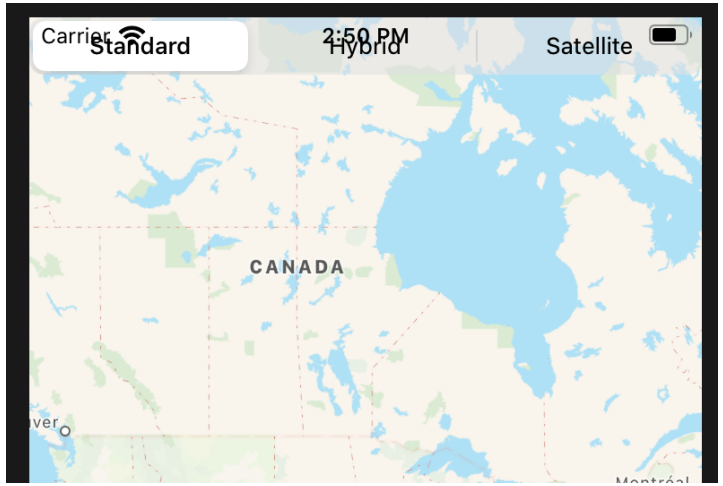
Making a constraint between to views active causes the constraint to be added to the view that is the nearest common ancestor between the views



Example: when a constraint between the two blue views is made active, it is added to the green view

Layout Guides

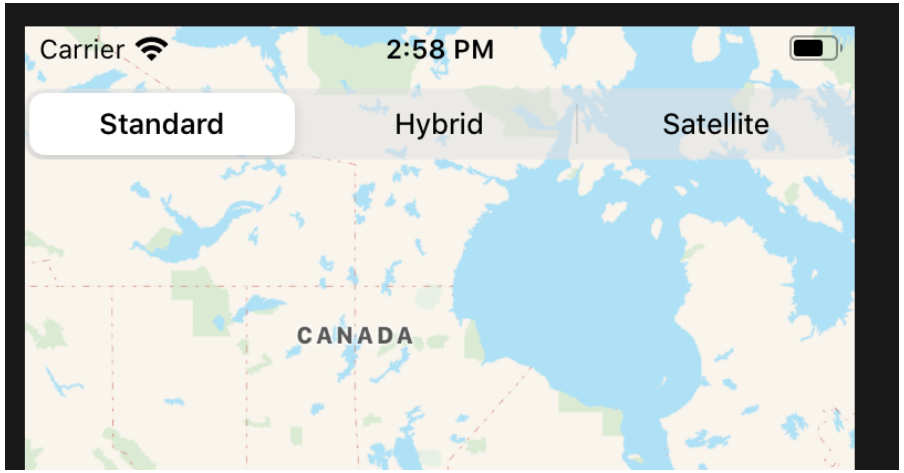
- Here is the segmented view, with the constraints in place
- Still not quite right—its crowded underneath the status bar
- A view controller exposes two properties that we can use for vertical constraints
  - `topLayoutGuide`: allows us to keep views out of the status bar (or navigation bar) at the top of the screen
  - `bottomLayoutGuide`: allows us to keep views out of the tab bar at the bottom of the screen
- The layout guides in turn expose three anchors that we can use
  - `topAnchor, bottomAnchor, heightAnchor`



Layout Guides

Code to use the `topLayoutGuide`

```
let topConstraint =
    segmentedControl.topAnchor.constraint(equalTo:topLayoutGuide.bottomAnchor, constant: 8)
```



Margins

- Every view has a `layoutMargins` property that denotes the default spacing to use when layout out content
- The actual value will change based on the characteristics of the device
- There is also a property called `layoutMarginsGuide`, which exposes anchors that are tied to the `layoutMargins`
- Use the the property in this way:

```
let margins = view.layoutMarginsGuide
let leadingConstraint =
    segmentedControl.leadingAnchor.constraint(equalTo: margins.leadingAnchor)
let trailingConstraint =
    segmentedControl.trailingAnchor.constraint(equalTo: margins.trailingAnchor)
```

Programmatic Controls

- `UIControl`: a view that can take input
  - such as `UIButton` or `UISegmentedControl`
- The input is represented by a `UIControlEvents` constant
- Here are come common ones:
  - `UIControlEvents.touchDown`: a touch down on the control
  - `UIConrolEvents.touchUpInside`: touch down followed by touch up while still within the bounds of the control
  - `UIControlEvents.valueChanged`: a touch that causes the value of the control to change
  - `UIControlEvents.editingChanged`: a touch that causes an editing change for a `UITextField`
- For the segmented control, we will use `UIControlEvents.valueChanged`

Target-Action Pair

To associate an action with a UIControlEvents.valueChanged action for the segmented control:

```
segmentedControl.addTarget(self, action: #selector(MapViewController.mapTypeChanged(_:)), for: .valueChanged)

@objc
func mapTypeChanged(_ segControl: UISegmentedControl) {
    switch segControl.selectedSegmentIndex {
    case 0:
        mapView.mapType = .standard
    case 1:
        mapView.mapType = .hybrid
    case 2:
        mapView.mapType = .satelliteFlyover
    default:
        break
    }
}
```

two things:

1. `#selector()` is required for the interface between Swift and Objective-C
2. `@objc` tells Objective-C about this method; can also decorate this with `@IBAction`

Target-Action Pair

Here's the code again:

```
segmentedControl.addTarget(self, action: #selector(MapViewController.mapTypeChanged(_:)), for: .valueChanged)
```

Another way of thinking about this:

- `UIControlEvents.valueChanged` is the type of event for which we are registering a listener
- `MapViewController.mapTypeChanged()` is the listener

Challenges

- Bronze challenge
  - follow the same steps you did for MapView
- Silver challenge
  - do some investigation
  - must change two values in Info.plist—easiest way is by editing Info.plist directly
- Gold challenge
  - here is some relevant information:

```
let UVMlat: Double = 44.0 + (28.0 + 33.0/60)/60
let UVMlon: Double = -(73.0 + (12.0 + 43.0/60)/60)
let UVMlocation = CLLocationCoordinate2D(latitude: UVMlat, longitude: UVMlon)
```