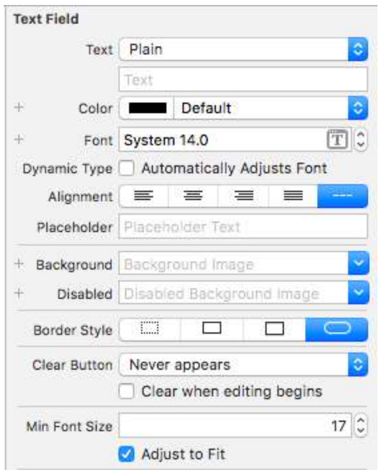
060010815:
iOS Application Development

Interface Design

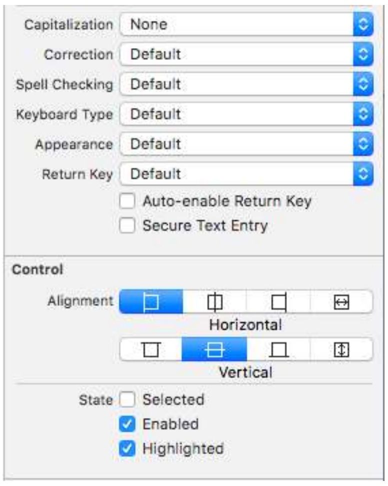
TextField

- A text field lets a user type in a single line of text

TextField


Text Field

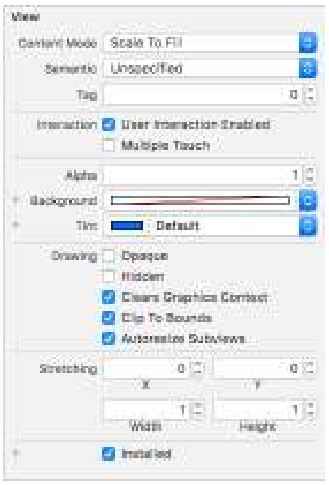
Text: Plain
Text
Color: Default
Font: System 14.0
Dynamic Type: ☐ Automatically Adjusts Font
Alignment
Placeholder: Placeholder Text
Background: Background Image
Disabled: Disabled Background Image
Border Style
Clear Button: Never appears
☐ Clear when editing begins
Min Font Size: 17
✓ Adjust to Fit


Capitalization: None
Correction: Default
Spell Checking: Default
Keyboard Type: Default
Appearance: Default
Return Key: Default
☐ Auto-enable Return Key
☐ Secure Text Entry

Control

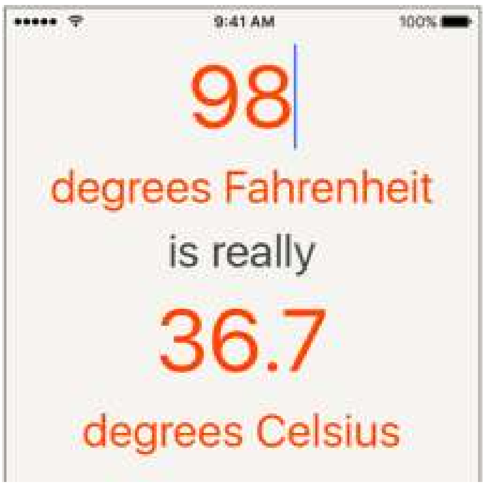
Alignment
Horizontal
Vertical
State: ☐ Selected
✓ Enabled
✓ Highlighted

TextField


View

Content Mode: Scale To Fill
Semantic: Unspecified
Tag: 0
Interaction: ✓ User Interaction Enabled
☐ Multiple Touch
Alpha: 1
Background
Tint: Default
Drawing: ☐ Opaque
☐ Hidden
✓ Clears Graphics Context
✓ Clip To Bounds
✓ Autoresize Subviews
Stretching: 0 0
X Y
1 1
Width Height
✓ Installed

TextField



98

degrees Fahrenheit

is really

36.7

degrees Celsius

TextField

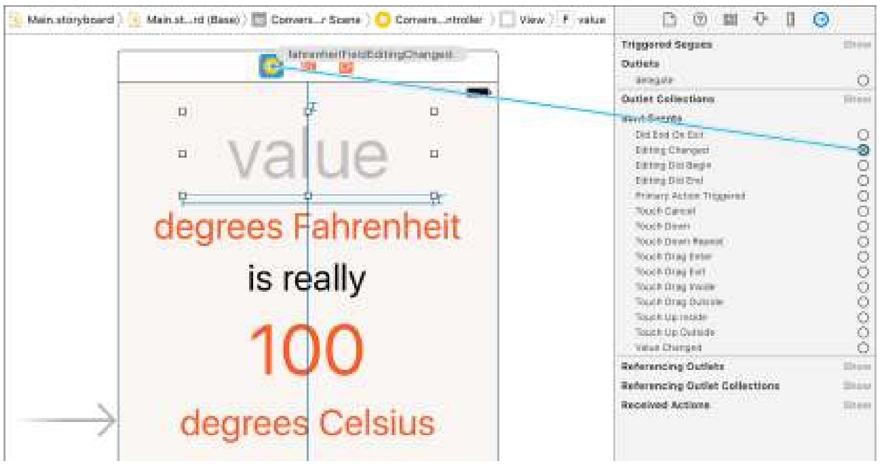
```
class ConversionViewController: UIViewController {

    @IBOutlet var celsiusLabel: UILabel!

    @IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {
        celsiusLabel.text = textField.text
    }
}
```

TextField

- Control-drag from the Conversion View Controller to the Celsius label
- Select the text field and open its connections inspector
- Click and drag from the circle to the right of Editing Changed to the Conversion View Controller and click the fahrenheitFieldEditingChanged

TextField

- Connecting the editing changed event



TextField

```
@IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {
    ~~celsiusLabel.text = textField.text~~

    if let text = textField.text, !text.isEmpty {
        celsiusLabel.text = text
    } else {
        celsiusLabel.text = "???"
    }
}
```

Dismissing the keyboard

- Currently, there is no way to dismiss the keyboard
- When the text field is tapped, the method **becomeFirstResponder()** is called on it.
- To dismiss the keyboard, call the method **resignFirstResponder()** on the text field.

Dismissing the keyboard

```
@IBOutlet var celsiusLabel: UILabel!
@IBOutlet var textField: UITextField!
```

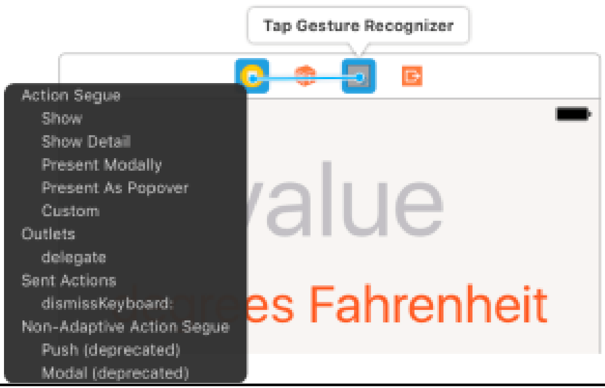
```
@IBAction func dismissKeyboard(_ sender: UITapGestureRecognizer) {
    textField.resignFirstResponder()
}
```

Dismissing the keyboard

- In Main.storyboard, find “**Tap Gesture Recognizer**” in the object library.
- Drag this object onto the background view for the View Controller.

Dismissing the keyboard

- Control-drag from the gesture recognizer in the scene dock to the View Controller and connect it to the dismissKeyboard: method



Dismissing the keyboard

- add a property for the Fahrenheit value

- @IBOutlet var celsiusLabel: UILabel!
- **var fahrenheitValue: Measurement<UnitTemperature>?**

Dismissing the keyboard

- Now add a computed property for the Celsius value.

```
var fahrenheitValue: Measurement<UnitTemperature>?

var celsiusValue: Measurement<UnitTemperature>? {
    if let fahrenheitValue = fahrenheitValue {
        return fahrenheitValue.converted(to: .celsius)
    } else {
        return nil
    }
}
```

Dismissing the keyboard

- Add a method to **ConversionViewController** that updates the celsiusLabel.

```
func updateCelsiusLabel() {
    if let celsiusValue = celsiusValue {
        celsiusLabel.text = "\(celsiusValue.value)"
    } else {
        celsiusLabel.text = "???"
    }
}
```

Dismissing the keyboard

- You want this method to be called whenever the Fahrenheit value changes.
- To do this, you will use a ***property observer***, which is a chunk of code that gets called whenever a property's value changes.

Dismissing the keyboard

- A property observer is declared using curly braces immediately after the property declaration.
- Inside the braces, you declare your observer using either **willSet** or **didSet**, depending on whether you want to be notified **immediately before** or **immediately after** the property value changes, respectively.

Dismissing the keyboard

- Add a property observer to fahrenheitValue that gets called after the property value changes.

```
var fahrenheitValue: Measurement<UnitTemperature>? {
    didSet {
        updateCelsiusLabel()
    }
}
```

Dismissing the keyboard

```
@IBAction func fahrenheitFieldEditingChanged(_ textField: UITextField) {

    ~~if let text = textField.text, !text.isEmpty {~~
        ~~celsiusLabel.text = text~~
    ~~} else {~~
        ~~celsiusLabel.text = "???"~~
    ~~}~~

    if let text = textField.text, let value = Double(text) {
        fahrenheitValue = Measurement(value: value, unit: .fahrenheit)
    } else {
        fahrenheitValue = nil
    }
}
```

Dismissing the keyboard

- Override **viewDidLoad()** to set the initial value

```
override func viewDidLoad() {
    super.viewDidLoad()

    updateCelsiusLabel()
}
```

Number formatters

- Use a *number formatter* to customize the display of a number

```
let numberFormatter: NumberFormatter = {
    let nf = NumberFormatter()
    nf.numberStyle = .decimal
    nf.minimumFractionDigits = 0
    nf.maximumFractionDigits = 1
    return nf
}()
```

Number formatters

- Now modify **updateCelsiusLabel()** to use this formatter.

```
func updateCelsiusLabel() {
    if let celsiusValue = celsiusValue {
        ~~celsiusLabel.text = "\(celsiusValue.value)"~~
        celsiusLabel.text =
            numberFormatter.string(from: NSNumber(value: celsiusValue.value))
    } else {
        celsiusLabel.text = "???"
    }
}
```

Delegation

- Delegation is an object-oriented approach to ***callbacks***.

- A callback is a function that is supplied in advance of an event and is called every time the event occurs.

Delegation

- When the user types into a text field, that text field will ask its delegate if it wants to accept the changes that the user has made.

- class ConversionViewController: UIViewController, **UITextFieldDelegate** {

Delegation

- Set the delegate property

```
func textField(_ textField: UITextField,
               shouldChangeCharactersIn range: NSRange,
               replacementString string: String) -> Bool {

    print("Current text: \(textField.text)")
    print("Replacement text: \(string)")

    return true
}
```

Delegation

```
func textField(_ textField: UITextField,
               shouldChangeCharactersIn range: NSRange,
               replacementString string: String) -> Bool {

    ~~print("Current text: \(textField.text)")~~
    ~~print("Replacement text: \(string)")~~

    ~~return true~~

    let existingTextHasDecimalSeparator = textField.text?.range(of: ".")
    let replacementTextHasDecimalSeparator = string.range(of: ".")

    if existingTextHasDecimalSeparator != nil,
        replacementTextHasDecimalSeparator != nil {
        return false
    } else {
        return true
    }
}
```

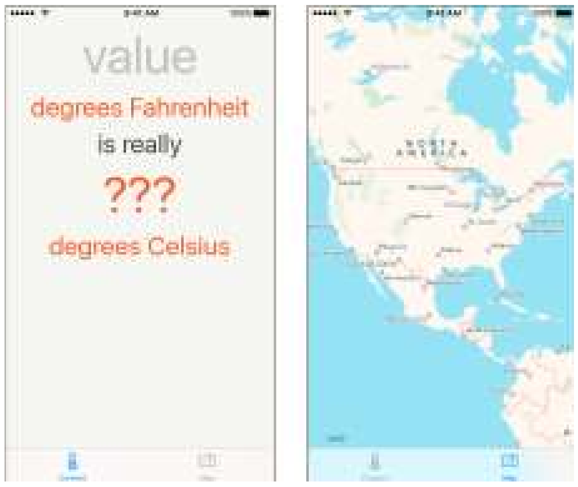
View Controller

- A view controller is
  - an instance of a subclass of **UIViewController**
  - manages a view hierarchy
  - responsible for creating view objects that make up the hierarchy
  - responsible for handling events associated with the view objects in its hierarchy

View Controller

- ***lazy loading***
  - view controller's view is not created until it needs to appear on the screen
  - conserve memory and improve performance
- view controller can create its view hierarchy:
  - in Interface Builder, by using an interface file such as a storyboard
  - programmatically, by overriding the **loadView()** method of **UIViewController**

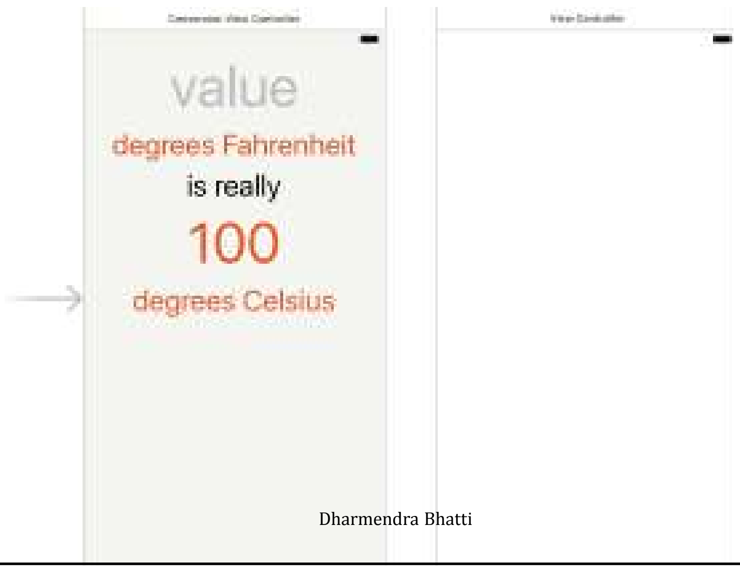
The two faces of WorldTrotter



Adding another view controller to the canvas

- Main.storyboard => object library => drag a View Controller onto the canvas

Adding another view controller to the canvas



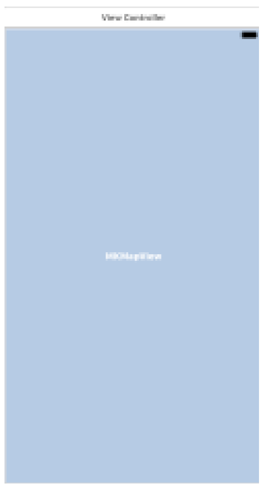
Replace UIView with MKMapView

- Select the view of the View Controller – not the View Controller itself! – and press Delete to remove this view from the canvas

- drag a Map Kit View from the object library onto the view controller

Dharmendra Bhatti 33

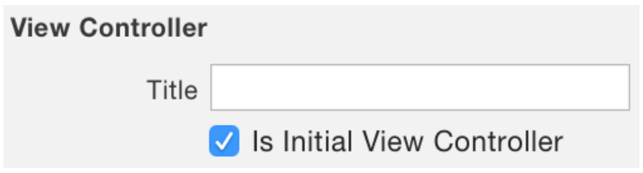
Replace UIView with MKMapView



Dharmendra Bhatti 34

Setting the initial view controller

- Now select the View Controller and open its attributes inspector.
- Under the View Controller section, check the box next to "Is Initial View Controller"


View Controller

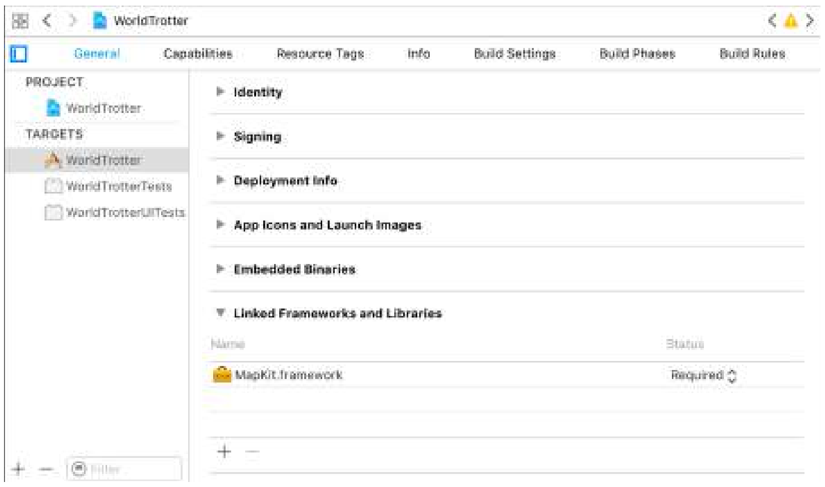
Title

[✓] Is Initial View Controller

Adding the MapKit framework

- Project Navigator =>
  - WorldTrotter Project Settings =>
    - Linked Frameworks and Libraries =>
      - Click on the + at the bottom and add MapKit.framework

Adding the MapKit framework


WorldTrotter

General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

PROJECT
WorldTrotter
TARGETS
WorldTrotter
WorldTrotterTests
WorldTrotterUITests

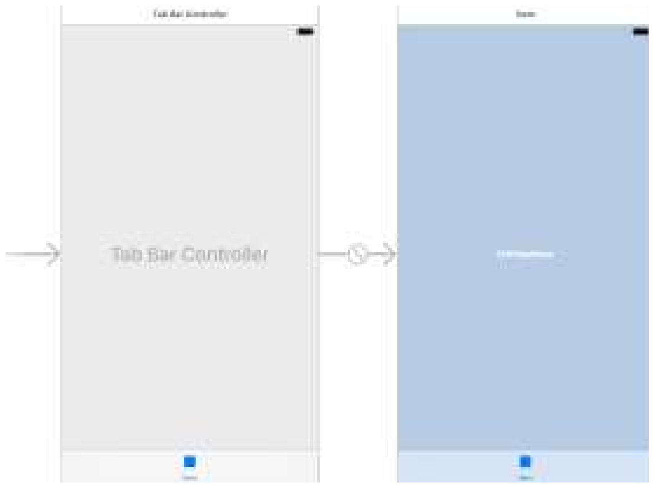
▸ Identity
▸ Signing
▸ Deployment Info
▸ App Icons and Launch Images
▸ Embedded Binaries
▾ Linked Frameworks and Libraries

Name	Status
MapKit.framework	Required

UITabBarController

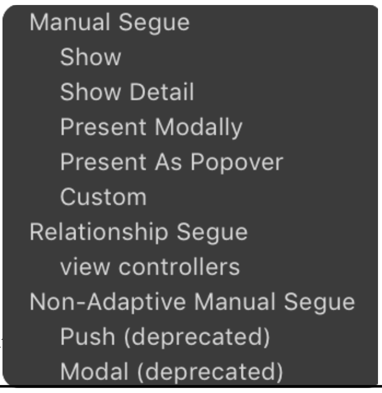
- Open Main.storyboard and select the View Controller.
- From the Editor menu, choose Embed In → Tab Bar Controller.

UITabBarController



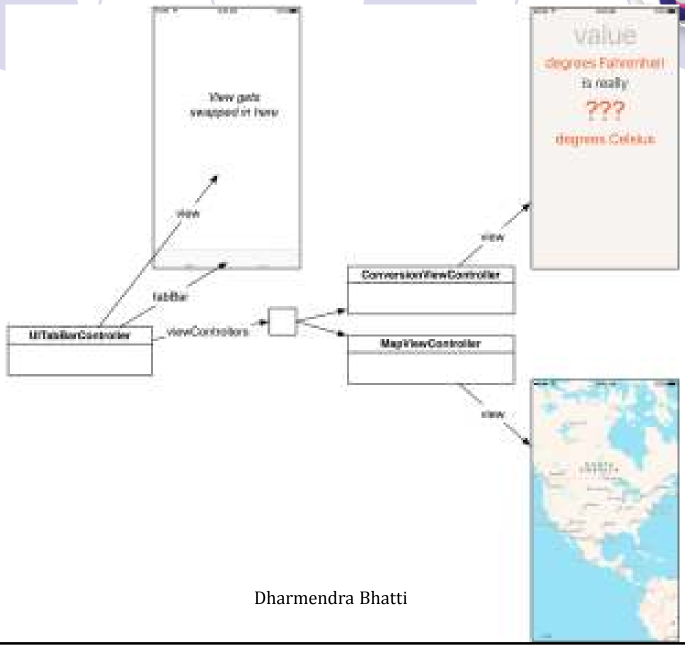
Dharmendra Bhatti 39

- Add the Conversion View Controller to the Tab Bar Controller's view controllers array.
- Control-drag from the Tab Bar Controller to the Conversion View Controller.
- From the Relationship Segue section, choose view controllers


Manual Segue
Show
Show Detail
Present Modally
Present As Popover
Custom
Relationship Segue
view controllers
Non-Adaptive Manual Segue
Push (deprecated)
Modal (deprecated)

Dharmendra Bha

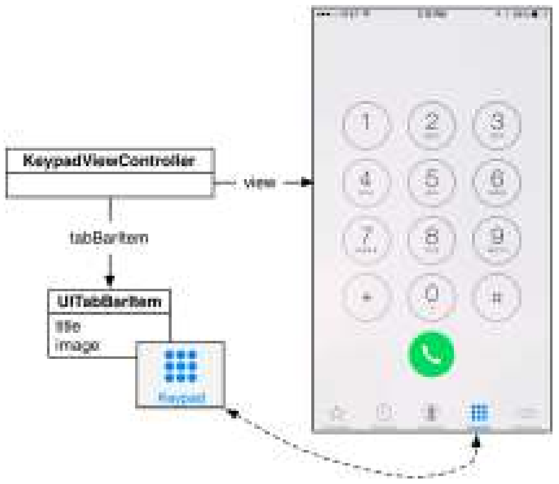
UITabBarController diagram



Tab bar items

- Each tab on the tab bar can display a **title** and an **image**
- Each view controller maintains a **tabBarItem** property

Tab bar items

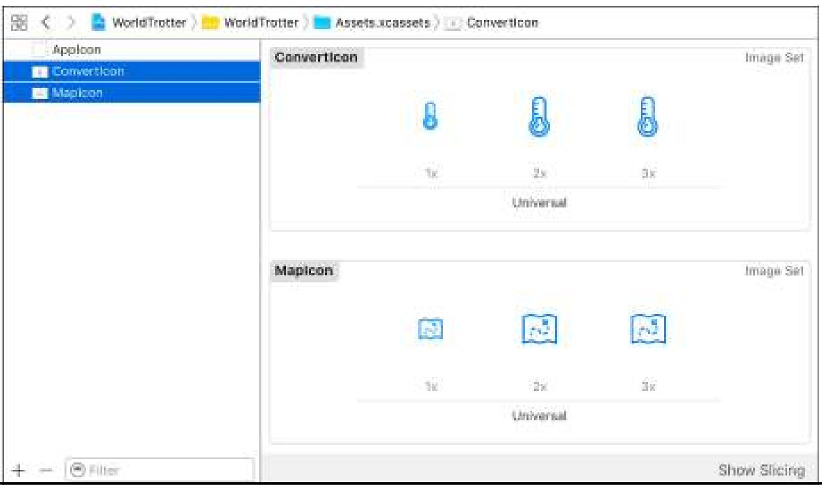


Tab bar items

- Adding images to the Asset Catalog
  - Drag ConvertIcon.png, ConvertIcon@2x.png, ConvertIcon@3x.png, MapIcon.png, MapIcon@2x.png, and MapIcon@3x.png files into the images set list on the left side of the Asset Catalog

Tab bar items

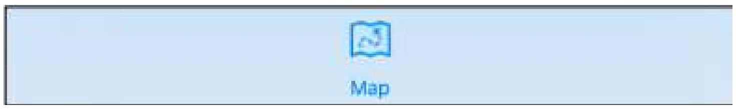
- Adding images to the Asset Catalog



45

Tab bar items

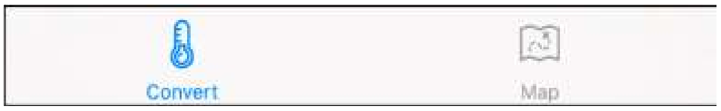
- Select Main.storyboard => View Controller => tab bar item
- Open attributes inspector
  - change the Title to “Map” and
  - choose MapIcon from the Image menu



Dharmendra Bhatti 46

Tab bar items

- Select Main.storyboard => Convert View Controller => tab bar item
- Open attributes inspector
  - change the Title to “Convert” and
  - choose ConvertIcon from the Image menu
  - change the first tab to be the Convert View Controller by dragging the tabs at the bottom of the Tab Bar Controller



Loaded and Appearing View

- The Lazy Loading
  - DO NOT load a view till it is needed (test this by writing code in **viewDidLoad()**)
  - tab bar controller defaults to loading the view of the first view controller in its array, which is the **ConvertViewController**

Add swift file for Map View Controller

- Create a new Swift file (Command-N) and name it MapViewController.
- Open MapViewController.swift and define a **UIViewController** subclass named **MapViewController**.

- import UIKit
- class MapViewController: UIViewController {
- }

Add swift file for Map View Controller

- Select Main.storyboard => Map View Controller => Identity Inspector => change the Class to MapViewController

Loaded and Appearing View

- In ConversionViewController.swift, update **viewDidLoad()**

```
override func viewDidLoad() {
    super.viewDidLoad()

    print("ConversionViewController loaded its view.")

    updateCelsiusLabel()
}
```

- In MapViewController.swift, override **viewDidLoad()**

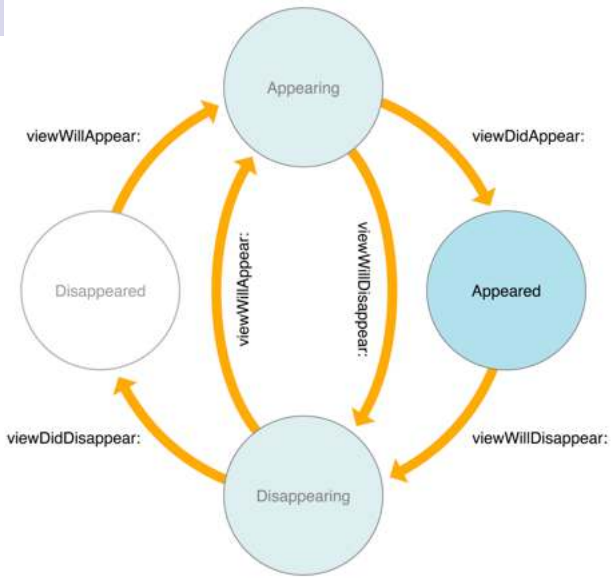
```
override func viewDidLoad() {
    super.viewDidLoad()

    print("MapViewController loaded its view.")
}
```

Loaded and Appearing View

- Override **viewDidLoad()** if the configuration only needs to be done once during the run of the app.

- Override **viewWillAppear(_:)** if you need the configuration to be done each time the view controller’s view appears onscreen.

Valid State Transitions



53

Interacting with View Controllers and Their Views

- **init(coder:)** is the initializer for **UIViewController** instances created from a storyboard.
- When a view controller instance is created from a storyboard, its **init(coder:)** gets called once.

Dharmendra Bhatti 54

Interacting with View Controllers and Their Views

- **init(nibName:bundle:)** is the designated initializer for **UIViewController**.
- When a view controller instance is created without the use of a storyboard, its **init(nibName:bundle:)** gets called once.
- This method will get called once on each view controller as it is created.

Interacting with View Controllers and Their Views

- **loadView()** is overridden to create a view controller's view programmatically.
- **viewDidLoad()** is overridden to configure views created by loading an interface file.
- This method gets called after the view of a view controller is created.

Interacting with View Controllers and Their Views

- **viewWillAppear(_:)** is overridden to configure views created by loading an interface file.

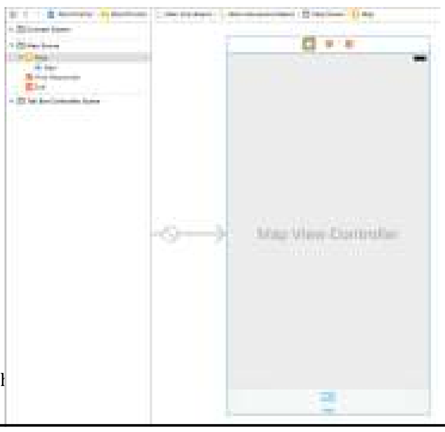
- This method and **viewDidAppear(_:)** get called every time your view controller is moved onscreen.

Interacting with View Controllers and Their Views

- **viewWillDisappear(_:)** and **viewDidDisappear(_:)** get called every time your view controller is moved offscreen.

Remove existing map view to create it programmatically

- In Main.storyboard, select the map view associated with Map View Controller and press Delete



Creating a View Programmatically

- If a view controller is asked for its view and its view is nil, then the **loadView()** method is called.

Creating a View Programmatically

```
import UIKit
import MapKit

class MapViewController: UIViewController {

    var mapView: MKMapView!

    override func loadView() {
        // Create a map view
        mapView = MKMapView()

        // Set it as *the* view of this view controller
        view = mapView
    }
```

Programmatic Constraints

- create and constrain your views in Interface Builder whenever possible.
- If your views are created in code, then you will need to constrain them programmatically

Segmented Control

- A segmented control allows the user to choose between a discrete set of options
- Programmatically add a **UISegmentedControl** to **MapViewController**’s interface (Hint: use loadView( ) method)

Segmented Control

- In MapViewController.swift, update loadView() to add segmented control to interface

```
override func loadView() {
    // Create a map view
    mapView = MKMapView()

    // Set it as *the* view of this view controller
    view = mapView

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

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

Programmatic Constraints

- Anchors
  - Anchors are properties on the view that correspond to attributes that you might want to constrain to an anchor on another view.
  - i.e. two views' leading edges being aligned.

Programmatic Constraints

- In MapViewController.swift, create these constraints in **loadView()**.

```
let topConstraint
        = segmentedControl.topAnchor.constraint(equalTo: view.topAnchor)
let leadingConstraint
        = segmentedControl.leadingAnchor.constraint(equalTo: view.leadingAnchor)
let trailingConstraint
        = segmentedControl.trailingAnchor.constraint(equalTo: view.trailingAnchor)
```

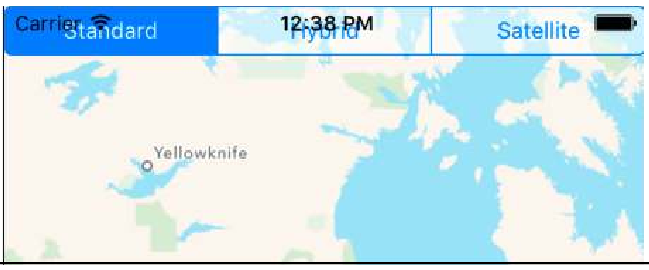
Activating Constraints

- topConstraint.isActive = true
- leadingConstraint.isActive = true
- trailingConstraint.isActive = true

Dharmendra Bhatti 67

Programmatic Constraints

- Build and run the application and switch to the **MapViewController**.
- The segmented control is now pinned to the top, leading, and trailing edges of its superview



68

Layout guides

- The layout guides indicate the extent to which the view controller's view contents will be visible.

Layout guides

- Using topLayoutGuide will allow your content to not overlap the status bar or navigation bar at the top of the screen.
- Using the bottomLayoutGuide will allow your content to not overlap the tab bar at the bottom of the screen.

Layout guides

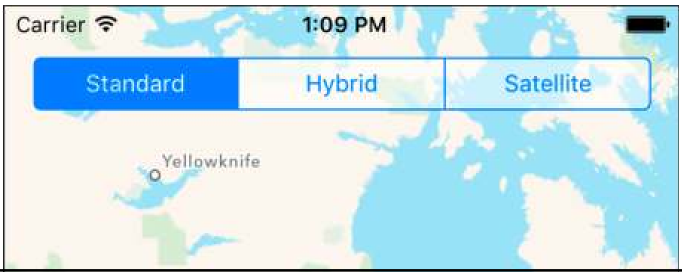
```
~~let topConstraint =~~
    ~~segmentedControl.topAnchor.constraint(equalTo: view.topAnchor)~~
let topConstraint =
    segmentedControl.topAnchor.constraint(equalTo: topLayoutGuide.bottomAnchor,
                                          constant: 8)
```

Margins

- Every view has a layoutMargins property that denotes the default spacing to use when laying out content.
- Advantage:
  - margins can change depending on the device type (iPad or iPhone) as well as the size of the device

Margins

```
let topConstraint =
    segmentedControl.topAnchor.constraint(equalTo: topLayoutGuide.bottomAnchor,
                                          constant: 8)
~~let leadingConstraint =~~
    ~~segmentedControl.leadingAnchor.constraint(equalTo: view.leadingAnchor)~~
~~let trailingConstraint =~~
    ~~segmentedControl.trailingAnchor.constraint(equalTo: view.trailingAnchor)~~

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



Programmatic Controls

- common control events

`UIControlEvents.touchDown`	A touch down on the control.
`UIControlEvents.touchUpInside`	A touch down followed by a 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`.

Programmatic Controls

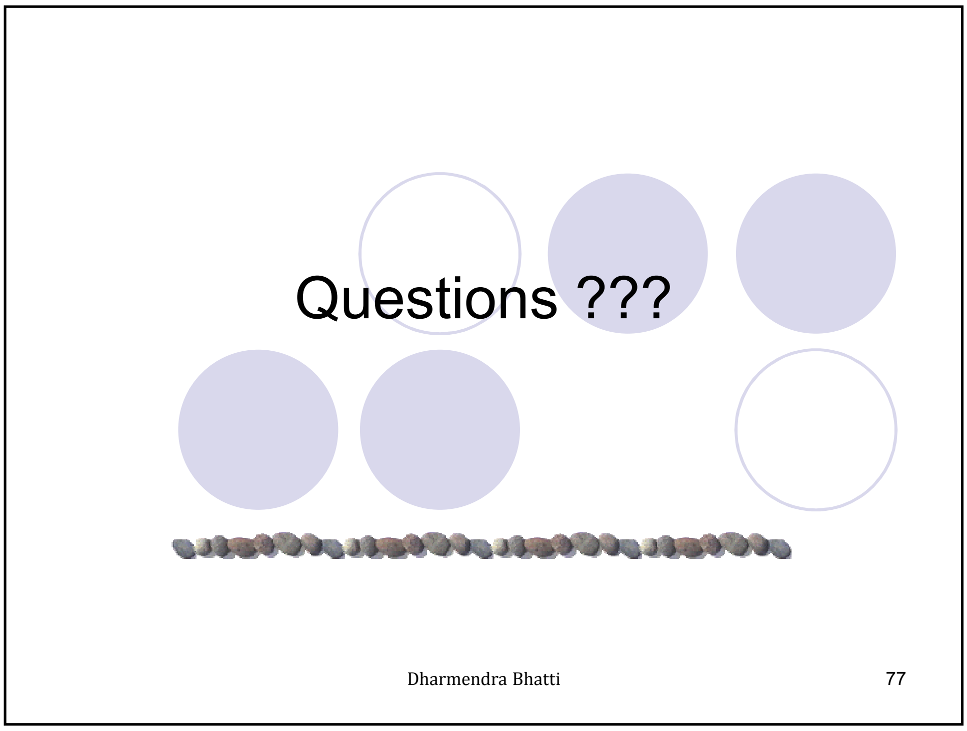
- In MapViewController.swift, update **loadView()** to add a target-action pair to the segmented control and associate it with the **.valueChanged** event.

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

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

Programmatic Controls

- Implement the action method **mapTypeChanged** in **MapViewController**

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



Questions ???

Dharmendra Bhatti

77