CSC 471 / 371 Mobile Application Development for iOS

Prof. Xiaoping Jia
School of Computing, CDM
DePaul University
xjia@cdm.depaul.edu
@DePaulSWEng

1

Outline

- Building user interface
- Basic controls
- Auto layout

DePaul University 2

Building UI – Part 1 Basic Controls

3

Components of an Application

- Compiled code
  - Your code: app delegate, view controllers, model classes, etc.
  - Frameworks: UIKit, Foundation, etc.
- Storyboard and xib files
  - `Main.storyboard`, `LaunchScreen.storyboard`, etc.
  - UI objects and other related objects, and their relationships
- Resources
  - images, audio/video, strings, etc.
  - `Info.plist` file (property list, in XML) – application configuration

DePaul University 4

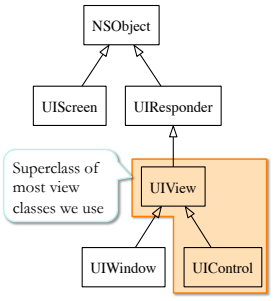
UIKit Framework

- Provides the *standard* user interface elements, view and control classes
  - Ready to use, customizable
  - Can be subclassed for more significant customizations: *custom view* classes
- Launches and manages your application
  - Follows well-defined patterns and conventions
- UIKit and you
  - Don't fight the frameworks
  - Understand the designs and how your code fit into them

DePaul University 5

User Interface Classes

- UI widgets are instances of classes that are subclasses of `UIView` or `UIControl`
- UI can be built
  - visually, using *Interface Builder*
  - or programmatically, in view controllers



DePaul University 6

Interface Builder and Storyboards

- Design time
  - *Interface Builder* helps you design the 'V' in MVC:
    - layout user interface objects
    - connect the elements in controllers (C) with view objects in UI (V)
  - UI objects, attributes, and relationships are *archived*, i.e., frozen, in a storyboard file
- At runtime, UI objects in a storyboard are *unarchived*
  - UI objects and attributes defined in the *Interface Builder* are restored
  - Ensures that the outlets and actions are connected
  - **Warning:** order of objects being unarchived is not defined

DePaul University 7

The View Controller – View Loading

```
import UIKit
class ViewController: UIViewController {
    @IBOutlet weak var label: UILabel!
    @IBAction func buttonPressed(sender: UIButton) {
        if let title = sender.currentTitle {
            label.text = "\(title) pressed"
        }
    }
    override func viewDidLoad() {
        super.viewDidLoad()
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
    }
}
```

Lifecycle callback

Override superclass method

DePaul University 64

View Loading in View Controllers

- View controllers initialization and *view loading* are managed by the system
- *View customization* and *unloading* are managed by apps
- *View loading* – Initializing and loading the view objects associated with the view controller into the memory
  - Views are **not** loaded when the a view controller is initialized
  - Views are loaded when it is accessed the first time, i.e., when it becomes visible.

DePaul University 66

View Loading in View Controllers

- If the views are defined in the storyboard, view loading is handled by the system automatically
  - View objects will be created and initialized
  - The view hierarchy will constructed
  - All outlets in the view controller will be connected to the corresponding view objects.
    - **Ensure not nil**. Declared as implicitly unwrapped optional types.
  - All actions in the view controller will be connected to the senders and the trigger events (target-action pattern)
- The `viewDidLoad` method is called after view loading has been completed to perform additional app specific customization.

DePaul University 67

Some UI Widgets in IB

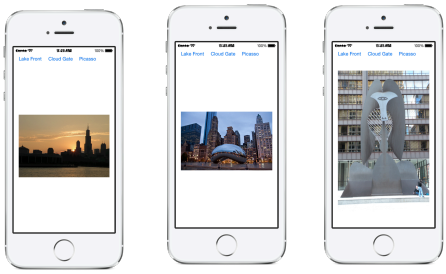
- Image View
  - a single image
- Label
  - read-only text
- Button
  - responds to touch
- Text Field/View
  - editable text
  - single/multi-line
- Switch
  - on/off state
- Slider
  - a range of values
- Segmented Control
  - multiple segments
- Stepper
  - inc/dec a value
- Table View
  - list of rows

DePaul University 8

Image Views Images – Chicago

9

Images – Chicago



DePaul University 10

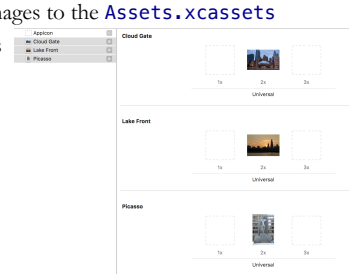
Start a New Project

- New project …
- iOS Application, Single View Application
  - Name: *Images – Chicago*
  - Language: Swift
  - Devices: iPhone
- In the `Main.storyboard`
  - Not suing auto layout
  - Design for *iPhone 4.7-inch*

DePaul University 11

Add Images to Assets Catalog

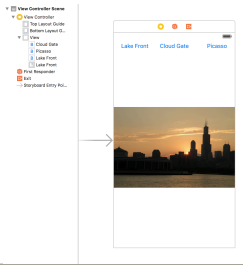
- Add three images to the `Assets.xcassets`
- Image names
  - Cloud Gate
  - Lake Front
  - Picasso



DePaul University 12

The Storyboard and UI Design

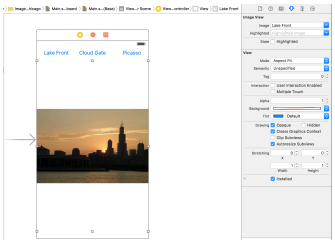
- Add three buttons, and an image view
- Button titles:
  - *Cloud Gate*
  - *Lake Front*
  - *Picasso*



DePaul University 13

Set Image View Attributes

- Image: *Lake Front*
- Mode: *Aspect Fit*



DePaul University 14

The Outlet and the Action

- Connect the image view to an outlet
  - Name: *image*
- Connect all three buttons to the same action
  - Name: *buttonPressed*

DePaul University 15

The View Controller

```
class ViewController: UIViewController {
    @IBOutlet weak var image: UIImageView!
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view,
    }
    @IBAction func buttonPressed(sender: UIButton) {
    }
    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```


The View Controller

```
class ViewController: UIViewController {
    @IBOutlet weak var image: UIImageView!

    @IBAction func buttonPressed(sender: UIButton) {
        image.image = UIImage(named: sender.currentTitle!)
    }
    …
}
```


The View Controller

```
class ViewController: UIViewController {
    @IBOutlet weak var image: UIImageView!

    @IBAction func buttonPressed(sender: UIButton) {
        if let name = sender.currentTitle {
            image.image = UIImage(named: name)
        }
    }
    …
}
```


Let's Try Some Different Devices

- iPhone 5s
- iPhone 6s
- iPhone 6s Landscape

Auto Layout

Adaptive Layout

- Making UI design adaptive
  - Dynamically adapt to *external* and *internal* size changes
  - Supporting any *size* display or *orientation* of a device
  - Great user experience for all situations
- Two separate mechanisms
  - Auto layout — This lecture
    - Constraint-based, dynamic layout approach
  - Size classes (handset vs tablet screen size) — Will discuss later
    - Allow different UI designs based on the *size classes* in each dimension, *horizontal* and *vertical*

© Xiaoping Jia, 2016

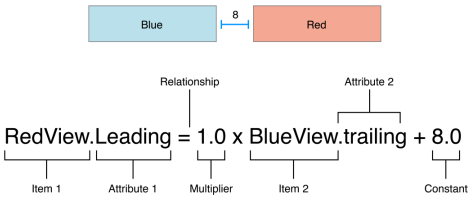
Auto Layout

- Automatically determine the size and location of each view object based on the screen size of the device, on which the app is running
  - Dynamically at runtime
- Constraint based
  - The location and size of each view object is defined based on a set of constraints
  - In relation to neighbors (siblings) or parent
- Using auto layout is the default for Xcode 7

Layout Constraints

- Each constraint is defined as a simple linear equation

$$x = a * y + b$$



Layout Constraints

- The layout of a view hierarchy is defined as a series of linear equations
  - *Efficient* algorithms available to solve the equations
- **Absence of constraint means unconstrained**
- It is possible to *under-* or *over-* constraint
  - Under-constraint: not enough constraints to fully determine the layout
    - Fix: add more constrains
  - Over-constraint: conflicts, it is not possible to satisfy all the constraints
    - Fix: remove some constrains, or adjust *priorities*

Define Layout Constraints

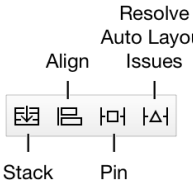
- Layout constraints can be defined in several different ways
- Graphically, in the *Interface Builder*
  - Specified at design time
  - Preview at design time
  - Resolved at runtime
- Programmatically, using `NSLayoutConstraint`
  - Run time, more flexible
- Textually, using the *Visual Format Language*
  - Run time, more flexible

Preferred method
Will cover here

Advanced topics

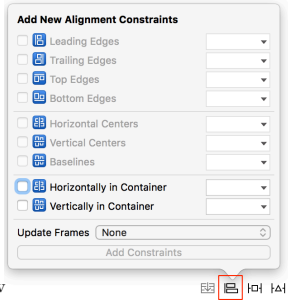
IB Tools for Auto Layout

- Four tools in the *Interface Builder*, located near the bottom
  - Stack
    - Create a stack view
  - Align
    - Align items in layout
  - Pin
    - Pin an item to its neighbor or parent
  - Resolve Auto Layout Issues
    - Fix common layout issues



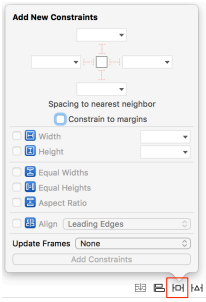
Auto Layout Constraints – Alignment

- The *Align* tool
- Align edges:
  - leading, trailing, top, bottom
  - Select multiple widgets: *Cmd-Click* or *Shift-Click*
- Align baseline
- Centering:
  - Horizontal or vertical
  - Container or neighbor view



Auto Layout Constraints – Space

- The *Pin* tool
- Width and height of widgets
- Distance to
  - container
  - neighboring view
- Equal width or height
  - Select multiple widgets: *Cmd-Click* or *Shift-Click*



Auto Layout Constraints – Attributes

- Each constraint has an optional *constant value*
  - Offset amount, distance, size, etc.
- Each constraint has a *priority*, 0 – 1000
  - To resolve conflicting constraints
    - High priority trumps low priority
- Leaf widgets, e.g., *Button* or *Label,* have *intrinsic content sizes*
  - Preferred size based on the content
  - Also define *priorities* for
    - Hugging contents, typically – mildly prefer tight hugging
    - Content compression resistance, typically – strongly resist compression

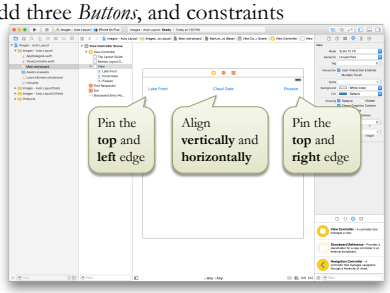
Auto Layout Issues

- The positions of widgets in IB is only a suggestion.
  - Actual position will be determined at run-time based on the constraints
  - You can preview, with different screen sizes
- Hints on auto layout in IB
  - Blue: the frame position in IB is *consistent* with the actual position at run-time
  - Orange: the frame position in IB is *inconsistent* with the actual position at run-time
    - Missing or inconsistent constraints
    - Need to the update the frame position

Image Views Images – Auto Layout

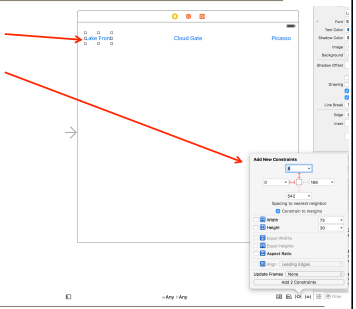
Add Layout Constraints

- Add three *Buttons*, and constraints



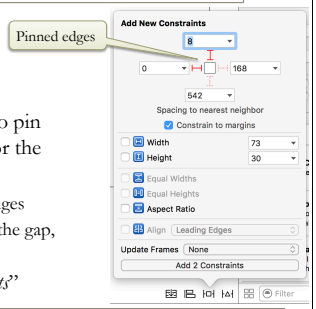
Add Pin Constraints

- Select the widget
- Use the *Pin* tool



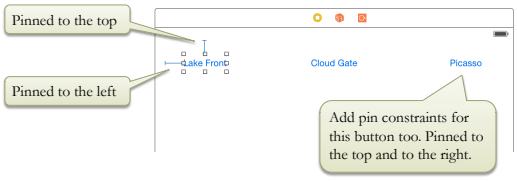
Add Pin Constraints

- Select the widget
- Use the *Pin* tool
- Select the *struts* of the corresponding edges to pin against the container or the nearest neighbor
  - e.g., the top and left edges
  - Adjust the values, i.e., the gap, if necessary
- Click "*Add 2 Constraints*"



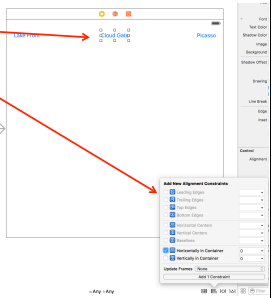
Add Pin Constraints

- After the pin constraints are added
  - The pin constraints are displayed as *struts* in the storyboard when the widget is selected



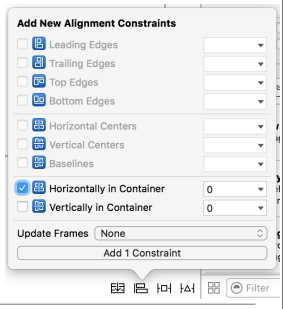
Add Alignment Constraints

- Select the widget
- Use the *Align* tool



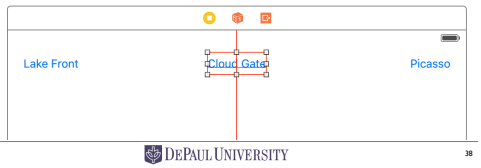
Add Alignment Constraints

- Select the widget
- Use the *Align* menu
- Check "*Horizontally in Container*"
- Click "*Add 1 Constraint*"



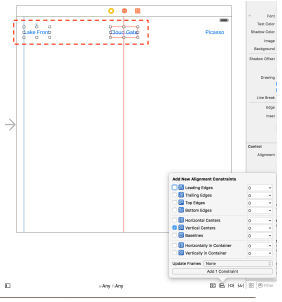
Add Alignment Constraints

- *"Houston, we have a problem!"*
  - **Orange colored lines**
- Missing a constraint:
  - The "Cloud Gate" button is constrained horizontally,
  - But, *not vertically*



Add Alignment Constraints

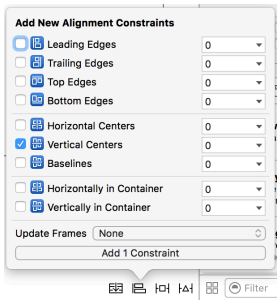
- Select the two widgets to be aligned
  - *Cmd-Click* or *Shift-Click*
- Use the *Align* tool



© Xiaoping Jia, 2016

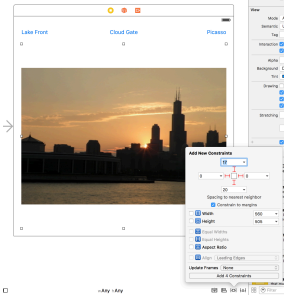
Add Alignment Constraints

- Select the two widgets to be aligned
  - *Cmd-Click* or *Shift-Click*
- Use the *Align* tool
- Check "*Vertical Centers*"
- Click "*Add 1 Constraint*"



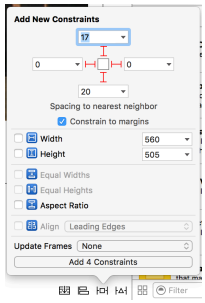
Add More Constraints

- Add an *Image View*
- Select the Image View
- Use the *Pin* tool



Add More Constraints

- Add an *Image View*
- Select the Image View
- Use the *Pin* tool
- Pin all four edges
  - Left, right, bottom to the container
  - Top to the nearest neighbor, i.e., the "Picasso" button
- Click "*Add 4 Constraints*"

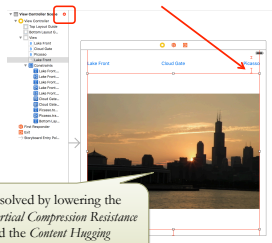


Layout Conflicts – Ambiguity

- If the view needs to be vertically compressed, which widget should be compressed?

  The *Button* or the *Image*?
- If the view needs to be vertically stretched, which widget should be stretched?

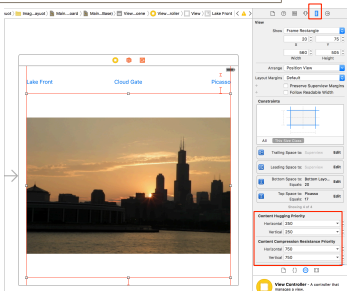
  The *Button* or the *Image*?



Resolved by lowering the *Vertical Compression Resistance* and the *Content Hugging* priority of the Image View

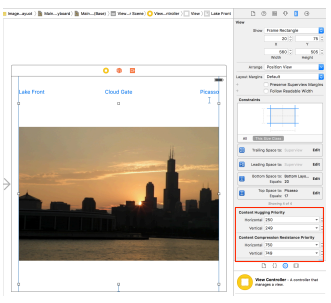
Set Constraint Priority

- Select the *Image View*
- Select the *Size Inspector*
- Find the "*Content Hugging*" priority
  - Default: 250
- Find the "*Content Compression Resistance*" priority
  - Default: 750



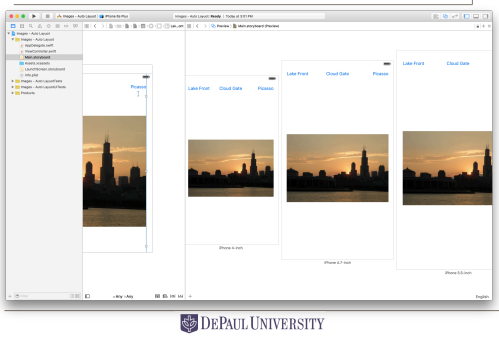
Set Constraint Priority

- Lower the "*Content Hugging*" priority
  - Default: 250
  - New value: 249
- Lower the "*Content Compression Resistance*" priority
  - Default: 750
  - New value: 749



© Xiaoping Jia, 2016

Preview Layout

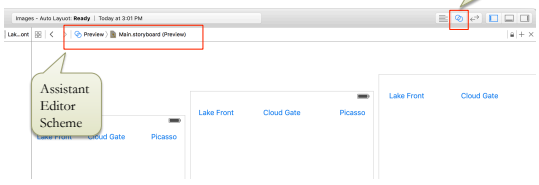


DePaul University 46

Preview Layout

- Select the *View Controller* of the scene to preview
- Show Assistant Editor
- Change the scheme from *Automatic* to *Preview*

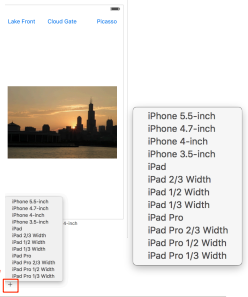
Show/hide Assistant Editor



DePaul University 47

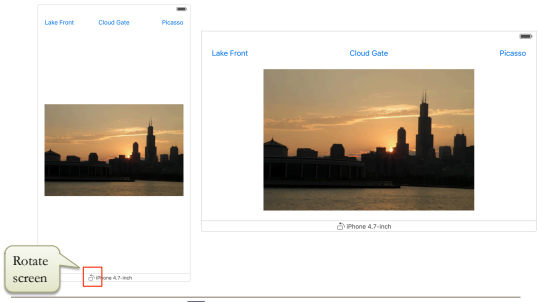
Preview Layout

- Add preview devices
  - Click the "+" at the lower left corner of the preview panel
- Available devices for preview
  - iPhone 3.5-inch
  - iPhone 4-inch
  - iPhone 4.7-inch
  - iPhone 5.5-inch
  - *etc.*



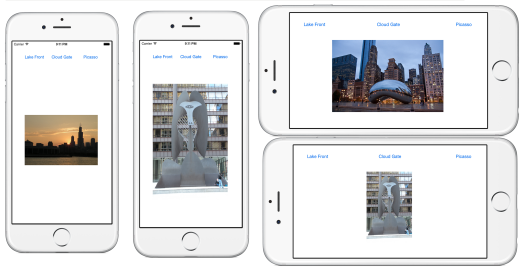
DePaul University 48

Preview Layout in Landscape



DePaul University 49

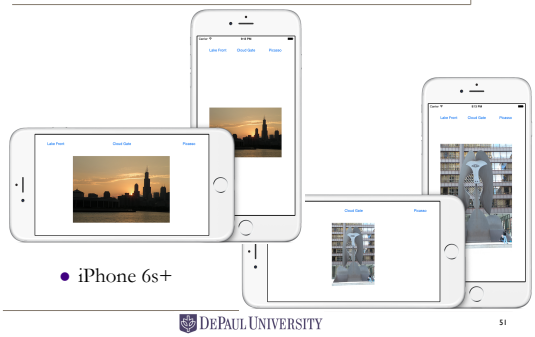
Running on Different Devices



- iPhone 6s
- iPhone 6s Landscape

DePaul University 50

Running on Different Devices



- iPhone 6s+

DePaul University 51

Control Widgets

Control Widgets

- `UISegmentedControl` 

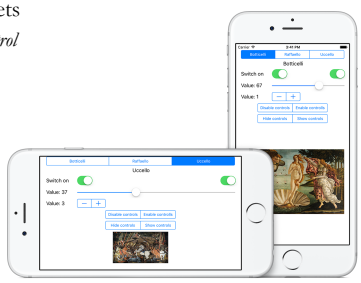
  - A horizontal group of buttons
  - Select one from the group, exclusive
- `UISwitch` 
  - Two-state, on/off button
- `UISlider`
  - Select a value from a continuous range (Float)
- `UIStepper` 
  - "+" and "–" buttons
  - Increment or decrement a value (Double)

Using Control Widgets

- Target-action pattern
  - Connect control widgets to actions in the view controller
  - Trigger event: *Value Changed*
- Properties to access the state of control widgets
  - `UISegmentedControl`
    `var selectedSegmentIndex: Int`
  - `UISwitch`
    `var on: Bool`
  - `UISlider`
    `var value: Double`
  - `UIStepper`
    `var value: Float`

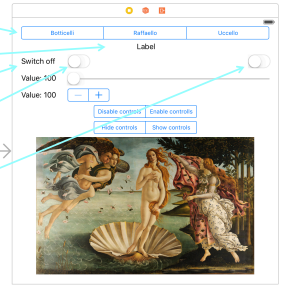
Controls Demo App

- Control widgets
  - *Segmented control*
  - *Switch*
  - *Slider*
  - *Stepper*
- Attributes
  - *enabled*
  - *hidden*



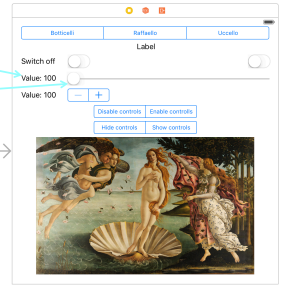
The Screen Layout

- Segmented Control
  - Pin top/left/right
- Name label
  - Pin top/left/right
- Switch label
  - Pin top/left
- Switch (left)
  - Pin top/left
- Switch (right)
  - Pin right
  - Align vertical center with the left switch



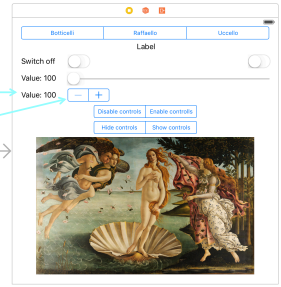
The Screen Layout

- Slider label
  - Pin top/left
- Slider
  - Align leading edge with left switch
  - Align trailing edge with right switch
  - Align vertical center with the slider label



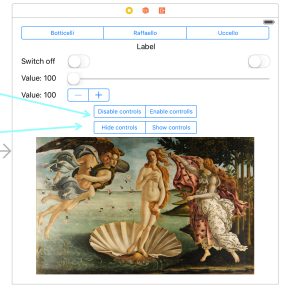
The Screen Layout

- Stepper label:
  - pin top/left
- Stepper
  - Align vertical center with the stepper label
  - Align leading edges with slider



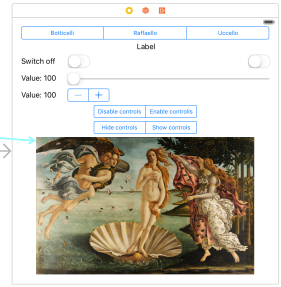
The Screen Layout

- Disable control
  - Pin top
  - Align center container
- Hidden control
  - pin top
  - Align leading/trailing edge with the disable control



The Screen Layout

- Image View
  - Pin top/left/right/bottom
  - Lower vertical *Content Compression Resistance Priority* to 749
  - Lower vertical *Content Hugging Priority* to 249



Connect the Outlets

```
class ViewController: UIViewController {
    @IBOutlet weak var nameLabel: UILabel!
    @IBOutlet weak var switchLabel: UILabel!
    @IBOutlet weak var sliderValue: UILabel!
    @IBOutlet weak var stepperValue: UILabel!

    @IBOutlet weak var image: UIImageView!

    @IBOutlet weak var leftSwitch: UISwitch!
    @IBOutlet weak var rightSwitch: UISwitch!
    @IBOutlet weak var slider: UISlider!
    @IBOutlet weak var stepper: UIStepper!
    …
}
```


Segmented Control Action

```
@IBAction func nameSelected(sender: UISegmentedControl) {
    var name = sender.titleForSegmentAtIndex(
                    sender.selectedSegmentIndex)
    nameLabel.text = name
    if let img = UIImage(named:name!) {
        image.image = img
    }
}
```

- Connected to the *Segmented Control* of names
  - Event: *Value Changed*
- Actions
  - Display the selected name and the corresponding image

Switch Action

```
@IBAction func switchToggled(sender: UISwitch) {
    switchLabel.text = "Switch " + (sender.on ? "on" : "off")
    leftSwitch.setOn(sender.on, animated: true)
    rightSwitch.setOn(sender.on, animated: true)
}
```

- Connected to both the left and right *Switches*
  - Event: *Value Changed*
- Action
  - Update text message *switch on* or *switch off*
  - Maintain synchrony of the left and right switches

© Xiaoping Jia, 2016

Slider Action

```
@IBAction func sliderMoved(sender: UISlider) {
    sliderValue.text = "Value: \(Int(sender.value))"
}
```

- Connected to the *Slider*
  - Event: *Value Changed*
- Action:
  - Display the current value of the slider

Stepper Action

```
@IBAction func stepperChanged(sender: UIStepper) {
    stepperValue.text = "Value: \(Int(sender.value))"
}
```

- Connected to the *Stepper*
  - Event: *Value Changed*
- Action:
  - Display the current value of the stepper

Disable/Enable Control Action

```
@IBAction func controlDisabled(sender: UISegmentedControl) {
    let enabled = (sender.selectedSegmentIndex == 1)
    leftSwitch.enabled = enabled
    rightSwitch.enabled = enabled
    slider.enabled = enabled
    stepper.enabled = enabled
}
```

- Connected to the *Segmented Control* for disabling and enabling controls
- Action:
  - Disable or enable the switches and the slider

Hide/Show Control Action

```
@IBAction func controlHidden(sender: UISegmentedControl) {
    let hidden = (sender.selectedSegmentIndex == 0)
    leftSwitch.hidden = hidden
    rightSwitch.hidden = hidden
    slider.hidden = hidden
    stepper.hidden = hidden
}
```

- Connected to the *Segmented Control* for hiding and showing controls
- Action:
  - Hide or show the switches and the slider

Initialize App State

- Lifecycle callback: `viewDidLoad`
  - Invoked *after* view loading is complete, and *before* the view is visible
- Opportunity to initialize the app state, before it is visible for the first time.
  - In our app: ensure the display labels reflect the actual value or state of the controls

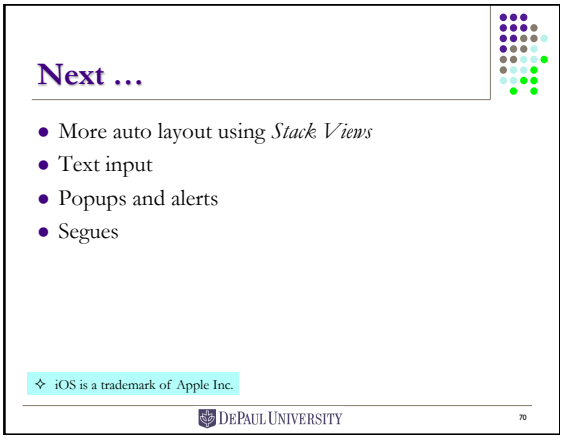
```
override func viewDidLoad() {
    super.viewDidLoad()

    nameLabel.text = "Botticelli"
    sliderValue.text = "Value: \(Int(slider.value))"
    stepperValue.text = "Value: \(Int(stepper.value))"
}
```


Sample Code

- Images – Chicago.zip
- Images – Auto Layout.zip
- Controls.zip

© Xiaoping Jia, 2016


Next …
More auto layout using *Stack Views*
Text input
Popups and alerts
Segues
iOS is a trademark of Apple Inc.
DePaul University
70