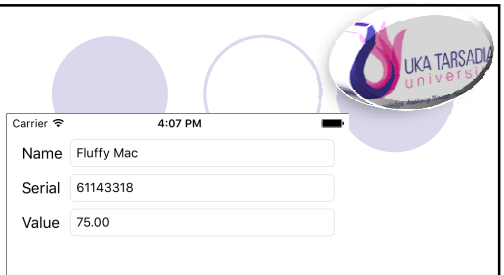
060010815:
iOS Application Development

Navigation, Touch and Gesture

Homepwner

- Homepwner with stack views



Aug 25, 2016

Homepwner

- Main.storyboard => View Controller from the object library onto the canvas.
- Drag a Vertical Stack View from the object library onto the view for the View Controller.
- Add constraints to the stack view to pin it to the leading and trailing margins, and pin the top and bottom edges to be 8 points from the top and bottom layout guides.

Homepwner

- Labels added to the stack view

Homepwner

- Date Created label => size inspector => set Vertical Content Hugging Priority to 249

Homepwner

- Repeat these steps for the Name, Serial, and Value labels:

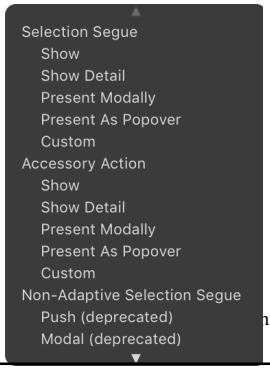
1. Select the label and click the ⊞ icon.
2. Change the stack view to be a horizontal stack view.
3. Drag a text field onto the horizontal stack view and change its horizontal content compression resistance priority to be 749.
4. Update the stack view to have a spacing of 8 points.

Homepwner

- vertical stack view => attributes inspector => set Spacing to 8 points.
- Date Created label => attributes inspector => change the Alignment to be centered
- Control-drag from the Name text field to the Serial text field and select Leading.
- Then do the same for the Serial text field and the Value text field.

Homepwner

- Main.storyboard => **ItemCell** prototype cell => Control-drag from the cell to the new view controller => Selection Segue => Show



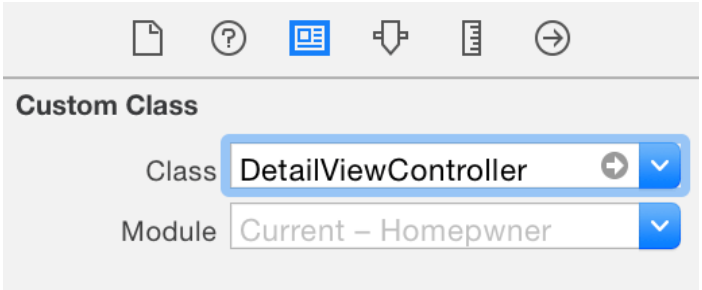
Homepwner

- Create a new Swift file and name it DetailViewController

```
import Foundation
import UIKit

class DetailViewController: UIViewController {

}
```


Homepwner

- Select the View Controller on the canvas and open its identity inspector.
- Change the Class to be DetailViewController



Homepwner

- Toggle the assistant editor by clicking the middle button from the Editor control at the top of the workspace.
- Command-Option-Return
  - The shortcut to display the assistant editor
- Command-Return
  - The shortcut to return to the standard editor

Homepwner

- Dragging from storyboard to source file

```
class DetailViewController: UIViewController {

    @IBOutlet var nameField: UITextField!
    @IBOutlet var serialNumberField: UITextField!
    @IBOutlet var valueField: UITextField!
    @IBOutlet var dateLabel: UILabel!

}
```

Homepwner

```
class DetailViewController: UIViewController {

    @IBOutlet var nameField: UITextField!
    @IBOutlet var serialNumberField: UITextField!
    @IBOutlet var valueField: UITextField!
    @IBOutlet var dateLabel: UILabel!

    var item: Item!

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated)

        nameField.text = item.name
        serialNumberField.text = item.serialNumber
        valueField.text = "\(item.valueInDollars)"
        dateLabel.text = "\(item.dateCreated)"
    }
}
```

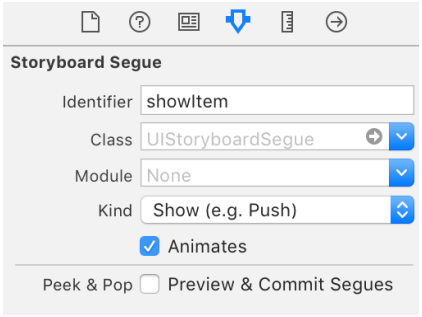

Homepwner

- Use number formatter

```
let numberFormatter: NumberFormatter = {
    let formatter = NumberFormatter()
    formatter.numberStyle = .decimal
    formatter.minimumFractionDigits = 2
    formatter.maximumFractionDigits = 2
    return formatter
}()

let dateFormatter: DateFormatter = {
    let formatter = DateFormatter()
    formatter.dateStyle = .medium
    formatter.timeStyle = .none
    return formatter
}()

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    nameField.text = item.name
    serialNumberField.text = item.serialNumber
    valueField.text = "\(item.valueInDollars)"
    dateLabel.text = "\(item.dateCreated)"
    valueField.text =
        numberFormatter.string(from: NSNumber(value: item.valueInDollars))
    dateLabel.text = dateFormatter.string(from: item.dateCreated)
}
```

Homepwner

- Main.storyboard again => click on the arrow between the two view controllers => open the attributes inspector => set identifier = showItem



15

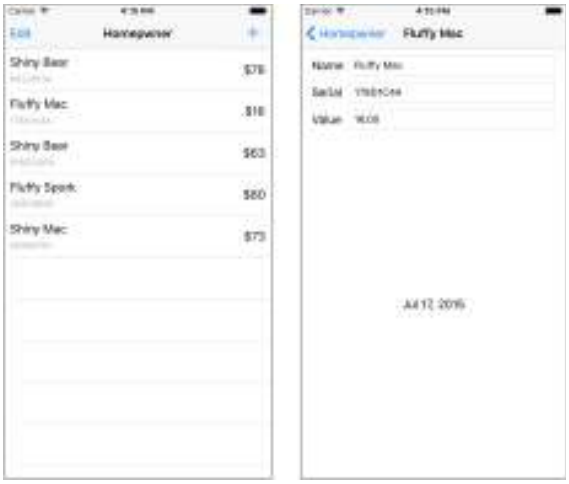
Homepwner

- ItemsViewController.swift

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // If the triggered segue is the "showItem" segue
    switch segue.identifier {
    case "showItem"?:
        // Figure out which row was just tapped
        if let row = tableView.indexPathForSelectedRow?.row {

            // Get the item associated with this row and pass it along
            let item = itemStore.allItems[row]
            let detailViewController
                    = segue.destination as! DetailViewController
            detailViewController.item = item
        }
    default:
        preconditionFailure("Unexpected segue identifier.")
    }
}
```

Drill-down interface in Settings

Homepwner with **UINavigationController**



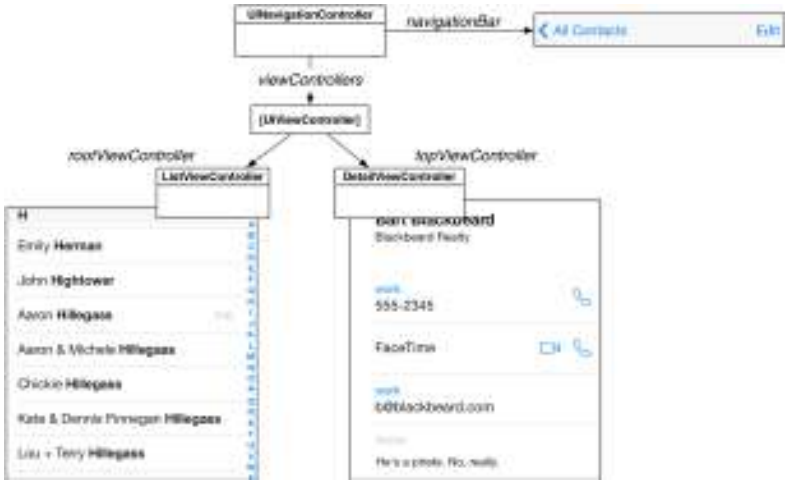
UINavigationController

- **UINavigationController**
  - is a subclass of **UIViewController**
  - maintains an array of view controllers presenting related information in a stack

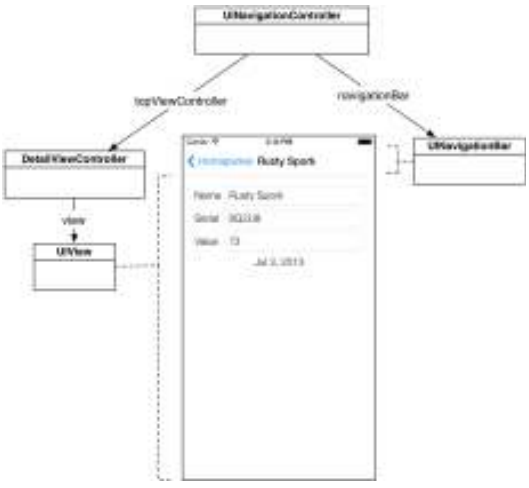
UINavigationController

- view controller is pushed onto the stack
  - view slides onscreen from the right
- When the stack is popped (i.e., the last item is removed),
  - the top view controller is removed from the stack
  - its view slides off to the right, exposing the view of the next view controller on the stack, which becomes the top view controller

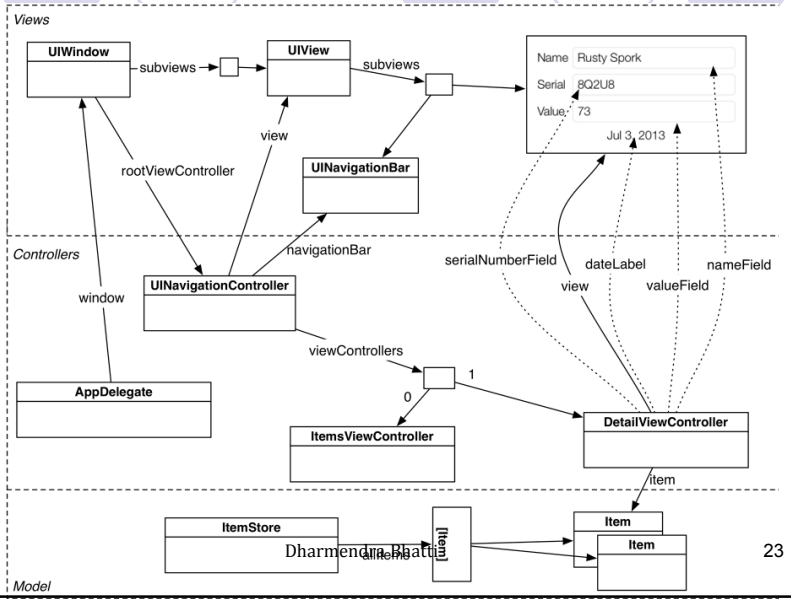
UINavigationController's stack



UINavigationController's view



Homepwner object diagram



Homepwner

- Main.storyboard => select the Items View Controller => Editor menu => Embed In → Navigation Controller

Homepwner

- AppDelegate.swift

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey : Any]?) -> Bool {
    // Override point for customization after application launch.

    // Create an ItemStore
    let itemStore = ItemStore()

    // Access the ItemsViewController and set its item store
    let itemsController = window!.rootViewController as! ItemsViewController
    let navController = window!.rootViewController as! UINavigationController
    let itemsController = navController.topViewController as! ItemsViewController
    itemsController.itemStore = itemStore

    return true
}
```


Homepwner

- In DetailViewController.swift

```
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    // "Save" changes to item
    item.name = nameField.text ?? ""
    item.serialNumber = serialNumberField.text

    if let valueText = valueField.text,
        let value = numberFormatter.number(from: valueText) {
        item.valueInDollars = value.intValue
    } else {
        item.valueInDollars = 0
    }
}
```

Homepwner

- values of the **Item** will be updated when the user taps the Back button on the **UINavigationBar**

- In ItemsViewController.swift

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    tableView.reloadData()
}
```


Homepwner

- Build and run the application.

Homepwner

- Dismissing the Keyboard
  - Dismissing by pressing the Return key
- class DetailViewController: UIViewController, **UITextFieldDelegate** {

```
func textFieldShouldReturn(_ textField: UITextField) -> Bool {
    textField.resignFirstResponder()
    return true
}
```


Homepwner

- open Main.storyboard and connect the delegate property of each text field to the Detail View Controller (Control-drag from each **UITextField** to the Detail View Controller and choose delegate.)

Homepwner

- Build and run the application.

Homepwner

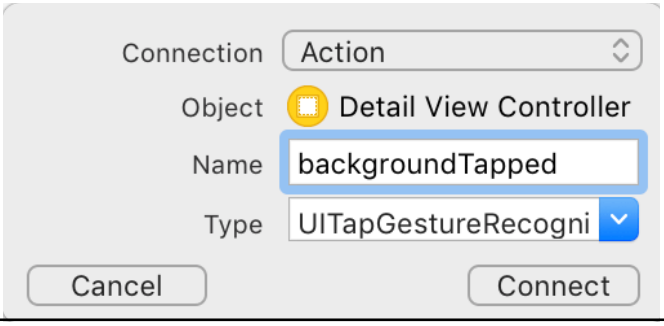
- Dismissing the Keyboard
  - Dismissing by tapping elsewhere
- Main.storyboard and find Tap Gesture Recognizer in the object library.
- Drag this object onto the background view for the Detail View Controller.

Homepwner

- In the project navigator, Option-click DetailViewController.swift to open it in the assistant editor.
- Control-drag from the tap gesture recognizer in the storyboard to the implementation of **DetailViewController**.

Homepwner

- select Action from the Connection menu.
- Name the action **backgroundTapped**.
- For the Type, choose UITapGestureRecognizer



Homepwner

- DetailViewController.swift
- @IBAction func backgroundTapped(_ sender: UITapGestureRecognizer) {
- **view.endEditing(true)**
- }

Homepwner

- Build and run the application.

Homepwner

- Dismiss the keyboard, when the user taps the Back button

```
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)

    // Clear first responder
    view.endEditing(true)
```

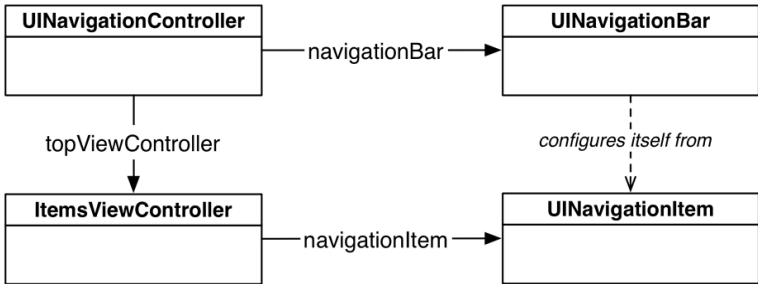
- ...
- ...

Homepwner

- Build and run the application.

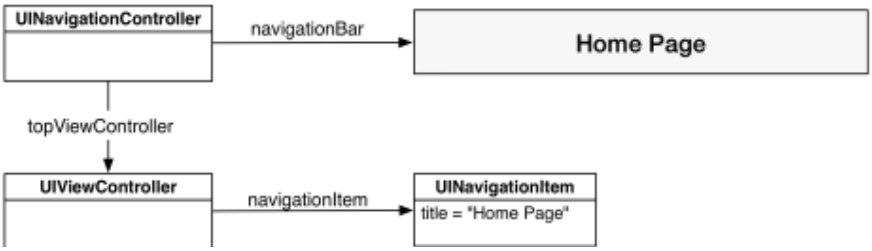
UINavigationItem

- Every **UIViewController** has a navigationItem property of type **UINavigationItem**



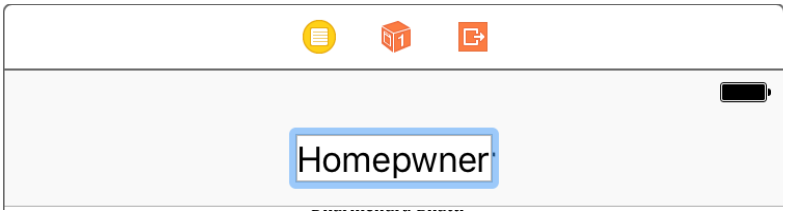
UINavigationItem

- **UINavigationItem** with title



UINavigationItem

- Open Main.storyboard.
- Double-click on the center of the navigation bar above the Items View Controller to edit its title.
- Give it a title of “Homepwner”



UINavigationItem

- In DetailViewController.swift, add a property observer to the item property that updates the title of the navigationItem.

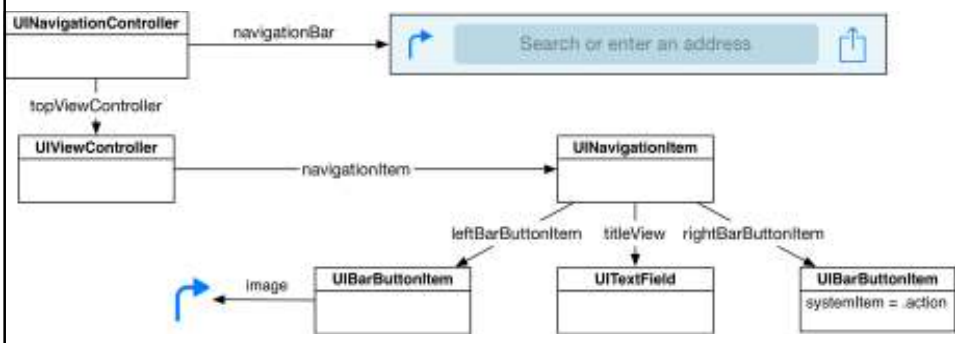
```
var item: Item! {
    didSet {
        navigationItem.title = item.name
    }
}
```

UINavigationItem

- Build and run the application.

UINavigationItem

- **UINavigationItem** with everything



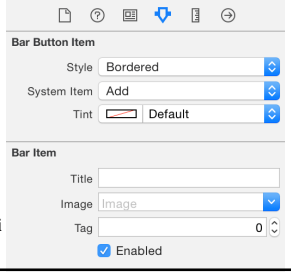
Adding buttons to the navigation bar

- In ItemsViewController.swift, update the method signature for **addNewItem(_:)**.

```
@IBAction func addNewItem(_ sender: UIButton) {
@IBAction func addNewItem(_ sender: UIBarButtonItem) {
    ...
}
```

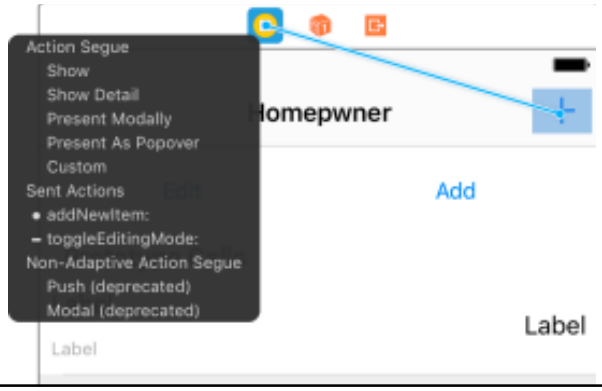

Adding buttons to the navigation bar

- Main.storyboard => object library => Drag a Bar Button Item to the right side of Items View Controller's navigation bar.
- Select this bar button item => attributes inspector => Change the System Item to Add



Connecting the addNewItem: action

- Control-drag from this bar button item to the Items View Controller and select addNewItem:



47

UINavigationItem

- Build and run the application.
- Tap the + button and a new row will appear in the table.

Add “Edit” Bar Button Item Programmatically

- In ItemsViewController.swift, override the **init(coder:)** method to set the left bar button item.

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)

    navigationItem.leftBarButtonItem = editButtonItem
}
```


UINavigationItem

- Build and run the application.

UINavigationItem

- The editButtonItem property creates a **UIBarButtonItem** with the title Edit.
- Even better, this button comes with a target-action pair: It calls the method **setEditing(_:animated:)** on its **UIViewController** when tapped.

Remove the header view and the associated code

- Select the header view on the table view and press Delete.
- In ItemsViewController.swift, delete the following code

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

    // Get the height of the status bar
    let statusBarHeight = UIApplication.shared.statusBarFrame.height

    let insets = UIEdgeInsets(top: statusBarHeight, left: 0, bottom: 0, right: 0)
    tableView.contentInset = insets
    tableView.scrollIndicatorInsets = insets

    tableView.rowHeight = UITableViewAutomaticDimension
    tableView.estimatedRowHeight = 65
}
```

Remove the header view and the associated code

- Finally, remove the **toggleEditingMode(_:)** method.

```
@IBAction func toggleEditingMode(_ sender: UIButton) {
    // If you are currently in editing mode...
    if isEditing {
        // Change text of button to inform user of state
        sender.setTitle("Edit", for: .normal)

        // Turn off editing mode
        setEditing(false, animated: true)
    } else {
        // Change text of button to inform user of state
        sender.setTitle("Done", for: .normal)

        // Enter editing mode
        setEditing(true, animated: true)
    }
}
```


Remove the header view and the associated code

- Build and run the application.

CAMERA

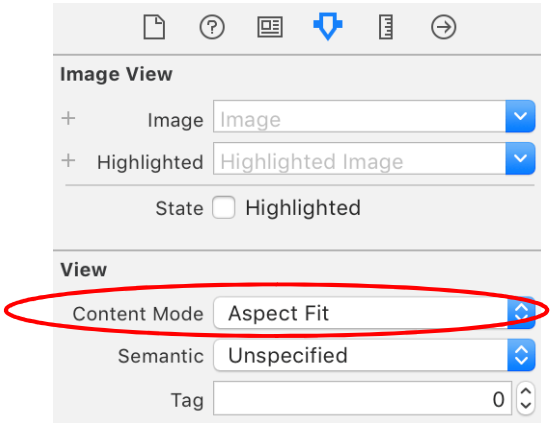
Homepwner with camera addition

Homepwner-UIImageView

- Homepwner.xcodeproj =>Main.storyboard => drag an instance of **UIImageView** onto the view at the bottom of the stack view
- Select image view => size inspector => lower Vertical Content Hugging Priority to be 248 and the Vertical Content Compression Resistance Priority to be 749

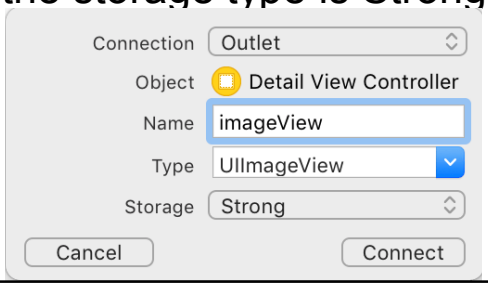
Homepwner-UIImageView

- Select UIImageView => attribute inspector => change “Content Mode” to “Aspect Fit”



Homepwner-UIImageView

- Option-click DetailViewController.swift
- Control-drag from the **UIImageView** to the top of DetailViewController.swift.
- Name the outlet imageView and make sure the storage type is Strong.



59

Homepwner-UIImageView

- DetailViewController.swift

```
class DetailViewController: UIViewController, UITextFieldDelegate {

    @IBOutlet var nameField: UITextField!
    @IBOutlet var serialNumberField: UITextField!
    @IBOutlet var valueField: UITextField!
    @IBOutlet var dateLabel: UILabel!
    @IBOutlet var imageView: UIImageView!
```

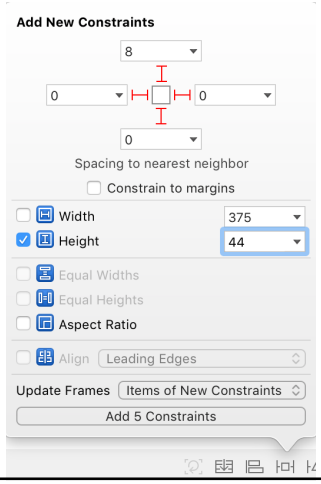
Adding a camera button

- Main.storyboard => Command-Return
- Select the bottom constraint for the stack view and press Delete to remove it.
- Drag the stack view up a bit

Dharmendra Bhatti

Adding a camera button

- Drag a toolbar from the object library onto the bottom of the view.
- Select the toolbar => Add New Constraints



62

Adding a camera button

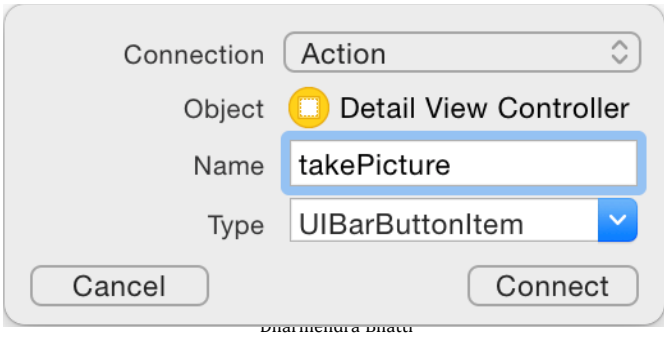
- Select UIToolBar => UIBarButtonItem => attribute inspector => Change the System Item to Camera

Adding a camera button

- Build and run the application

Adding a camera button

- Option-click DetailViewController.swift
- Control-drag from the camera button to DetailViewController.swift



Taking Pictures and UIImagePickerController

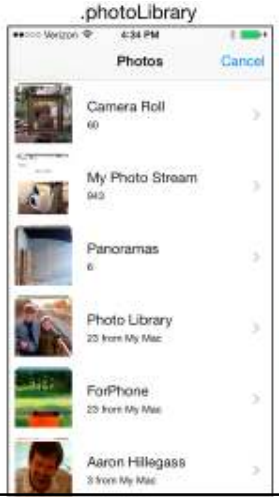
- Setting the image picker's sourceType
  - UIImagePickerControllerSourceType.**camera**
    - Allows the user to take a new photo.
  - UIImagePickerControllerSourceType.**photoLibrary**
    - Prompts the user to select an album and then a photo from that album.
  - UIImagePickerControllerSourceType.**savedPhotosAlbum**
    - Prompts the user to choose from the most recently taken photos.

Taking Pictures and UIImagePickerController

- UIImagePickerControllerSourceType

.camera

.photoLibrary



.savedPhotosAlbum

Taking Pictures and UIImagePickerController

- DetailViewController.swift

```
@IBAction func takePicture(_ sender: UIBarButtonItem) {

    let imagePicker = UIImagePickerController()

    // If the device has a camera, take a picture; otherwise,
    // just pick from photo library
    if UIImagePickerController.isSourceTypeAvailable(.camera) {
        imagePicker.sourceType = .camera
    } else {
        imagePicker.sourceType = .photoLibrary
    }
}
```

Setting the image picker's delegate

- class DetailViewController: UIViewController, UITextFieldDelegate, **UINavigationControllerDelegate, UIImagePickerControllerDelegate** {

Setting the image picker's delegate

- In DetailViewController.swift

```
@IBAction func takePicture(_ sender: UIBarButtonItem) {

    let imagePicker = UIImagePickerController()

    // If the device has a camera, take a picture; otherwise,
    // just pick from photo library
    if UIImagePickerController.isSourceTypeAvailable(.camera) {
        imagePicker.sourceType = .camera
    } else {
        imagePicker.sourceType = .photoLibrary
    }

    imagePicker.delegate = self
}
```

Presenting the image picker modally

- In DetailViewController.swift, add code to the end of **takePicture(_:)**

```
    imagePicker.delegate = self

    // Place image picker on the screen
    present(imagePicker, animated: true, completion: nil)
}
```


Permissions

- Application must supply a *usage description* that specifies the reason that your application wants to access particular information
  - Camera and photos
  - Location
  - Microphone
  - HealthKit data
  - Calendar
  - Reminders

Permissions

- In the project navigator, select the project at the top => open the Info tab along the top

Permissions

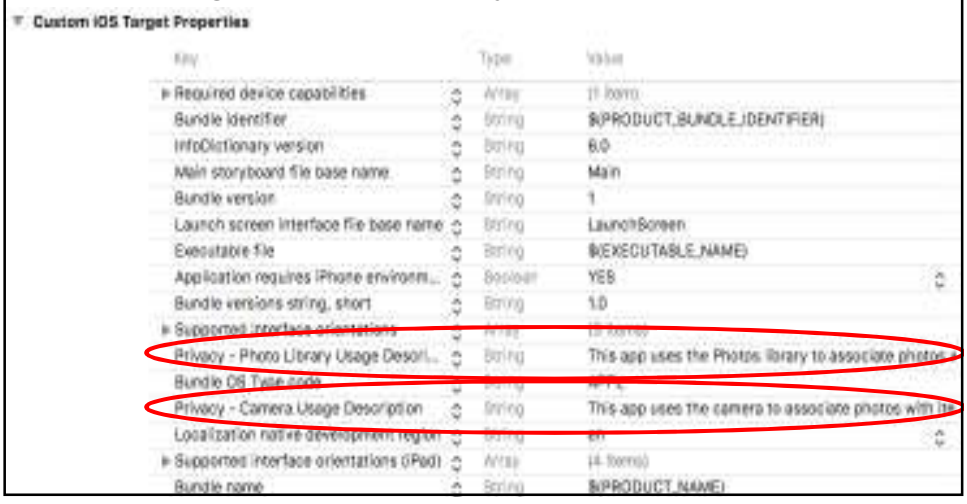
- Hover over the last entry in this list of Custom iOS Target Properties and click the + button
- Set the Key to be NSCameraUsageDescription
- Set value = “This app uses the camera to associate photos with items.”

Permissions

- Hover over the last entry in this list of Custom iOS Target Properties and click the + button
- Set the Key to be NSPhotoLibraryUsageDescription
- Set value = “This app uses the Photos library to associate photos with items.”

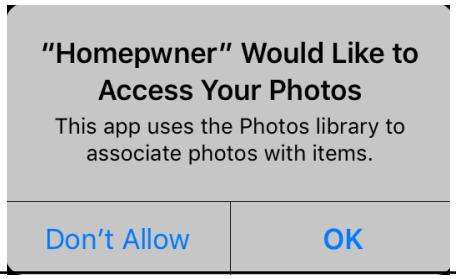
Permissions

- Adding in the new keys



Permissions

- Build and run the application and navigate to an item.
- Tap the camera button and you will see the permission dialog presented with the usage description



Saving the image

- In DetailViewController.swift,

```
func imagePickerController(_ picker: UIImagePickerController,
        didFinishPickingMediaWithInfo info: [String: Any]) {

    // Get picked image from info dictionary
    let image = info[UIImagePickerControllerOriginalImage] as! UIImage

    // Put that image on the screen in the image view
    imageView.image = image

    // Take image picker off the screen -
    // you must call this dismiss method
    dismiss(animated: true, completion: nil)
}
```

Saving the image

- Build and run the application again.
- Detail view => Select a photo.
- The image picker is dismissed, and you are returned to the **DetailViewController**'s view, where you will see the selected photo.

Creating ImageStore

- Create a new Swift file named ImageStore.

```
import Foundation
import UIKit

class ImageStore {

    let cache = NSCache<NSString,UIImage>()

}
```

Creating ImageStore

- Implement three methods in ImageStore class for adding, retrieving, and deleting an image from the dictionary

```
func setImage(_ image: UIImage, forKey key: String) {
    cache.setObject(image, forKey: key as NSString)
}

func image(forKey key: String) -> UIImage? {
    return cache.object(forKey: key as NSString)
}

func deleteImage(forKey key: String) {
    cache.removeObject(forKey: key as NSString)
}
```


Giving View Controllers Access to the Image Store

- In DetailViewController.swift,
  - **var imageStore: ImageStore!**
- In ItemsViewController.swift
  - **var imageStore: ImageStore!**

Giving View Controllers Access to the Image Store

- In ItemsViewController.swift,

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // If the triggered segue is the "showItem" segue"
    switch segue.identifier {
    case "showItem"?:
        // Figure out which row was just tapped
        if let row = tableView.indexPathForSelectedRow?.row {

            // Get the item associated with this row and pass it along
            let item = itemStore.allItems[row]
            let detailViewController
                    = segue.destination as! DetailViewController
            detailViewController.item = item
            detailViewController.imageStore = imageStore
        }
    default:
        preconditionFailure("Unexpected segue identifier.")
    }
}
```


Giving View Controllers Access to the Image Store

- In AppDelegate.swift,

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey: Any]?) -> Bool {
    // Override point for customization after application launch.

    // Create an ItemStore
    let itemStore = ItemStore()

    // Create an ImageStore
    let imageStore = ImageStore()

    // Access the ItemsViewController and set its item store and image store
    let navController = window!.rootViewController as! UINavigationController
    let itemsController = navController.topViewController as! ItemsViewControlle
    itemsController.itemStore = itemStore
    itemsController.imageStore = imageStore
```

Creating and Using Keys

- Add a property to Item.swift to store the key.
- **let itemKey: String**

Creating and Using Keys

- In Item.swift, generate a UUID and set it as the itemKey.

```
init(name: String, serialNumber: String?, valueInDollars: Int) {
    self.name = name
    self.valueInDollars = valueInDollars
    self.serialNumber = serialNumber
    self.dateCreated = Date()
    self.itemKey = UUID().uuidString

    super.init()
}
```

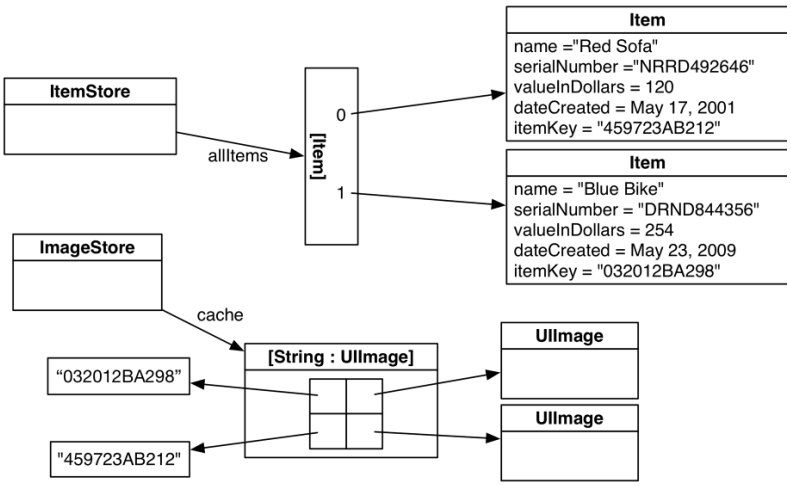
Creating and Using Keys

- In DetailViewController.swift,

```
func imagePickerController(_ picker: UIImagePickerController,
        didFinishPickingMediaWithInfo info: [String : Any]) {

    // Get picked image from info dictionary
    let image = info[UIImagePickerControllerOriginalImage] as! UIImage

    // Store the image in the ImageStore for the item's key
    imageStore.setImage(image, forKey: item.itemKey)

    // Put that image on the screen in the image view
    imageView.image = image

    // Take image picker off the screen -
    // you must call this dismiss method
    dismiss(animated: true, completion: nil)
}
```


Creating and Using Keys

- Accessing images from the cache



Creating and Using Keys

- When an item is deleted, delete its image from the image store

```
override func tableView(_ tableView: UITableView,
                        commit editingStyle: UITableViewCellEditingStyle,
                        forRowAt indexPath: IndexPath) {
    // If the table view is asking to commit a delete command...
    if editingStyle == .delete {
        let item = itemStore.allItems[indexPath.row]

        let title = "Delete \(item.name)?"
        let message = "Are you sure you want to delete this item?"

        let ac = UIAlertController(title: title,
                                   message: message,
                                   preferredStyle: .actionSheet)

        let cancelAction = UIAlertAction(title: "Cancel",
                                         style: .cancel,
                                         handler: nil)
        ac.addAction(cancelAction)

        let deleteAction = UIAlertAction(title: "Delete", style: .destructive,
                                         handler: { (action) -> Void in
            // Remove the item from the store
            self.itemStore.removeItem(item)

            // Remove the item's image from the image store
            self.imageStore.deleteImage(forKey: item.itemKey)
```


Wrapping Up ImageStore

- In DetailViewController.swift,

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    nameField.text = item.name
    serialNumberField.text = item.serialNumber
    valueField.text =
        numberFormatter.string(from: NSNumber(value: item.valueInDollars))
    dateLabel.text = dateFormatter.string(from: item.dateCreated)

    // Get the item key
    let key = item.itemKey

    // If there is an associated image with the item
    // display it on the image view
    let imageToDisplay = imageStore.image(forKey: key)
    imageView.image = imageToDisplay
}
```

Wrapping Up ImageStore

- Build and run the application.

TOUCH EVENTS

Touch Events

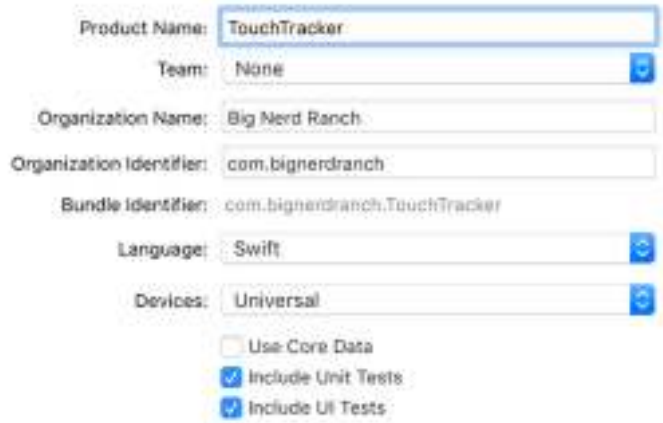
- one or more fingers touch the screen
  - **func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?)**
- one or more fingers move across the screen (this message is sent repeatedly as a finger moves)
  - **func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?)**

Touch Events

- one or more fingers are removed from the screen
  - **func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?)**
- a system event, like an incoming phone call, interrupts a touch before it ends
  - **func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?)**

Demo – TouchTracker

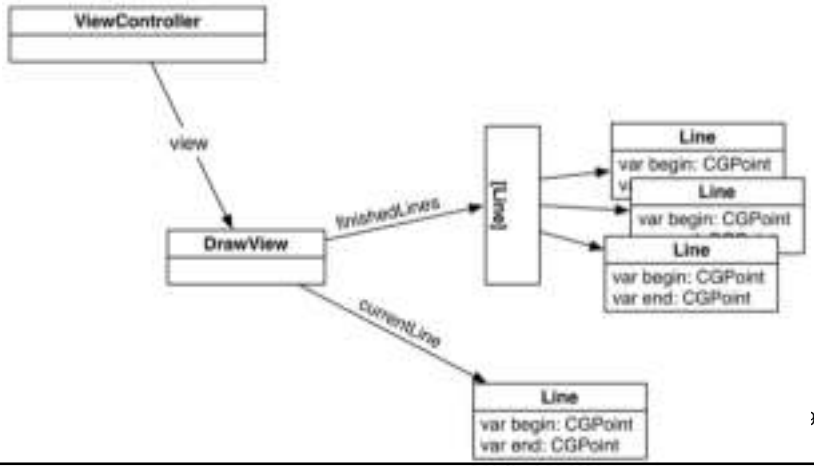
- Create a NEW project “TouchTracker”



95

Demo – TouchTracker

- Object diagram for TouchTracker



96

Demo – TouchTracker

- Create a new Swift file named Line
  - import Foundation
  - **import CoreGraphics**
  - **struct Line {**
  - **var begin = CGPoint.zero**
  - **var end = CGPoint.zero**
  - **}**

Structs

- Structs (and enums) are *value types* where as classes are *reference types.*
- Structs do not support inheritance.
- Structs don't have deinitializer

Structs

- Structs get a *member-wise initializer* if no other initializers are declared
  - **init(begin: CGPoint, end: CGPoint)**
- If all properties have default values and no other initializers are declared, structs also gain an empty initializer **init()**

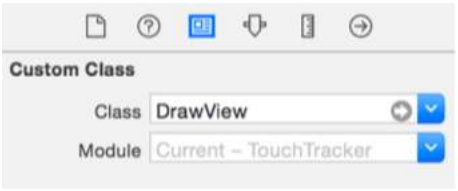
Demo – TouchTracker

- Create a new Swift file named DrawView

```
~~import Foundation~~
import UIKit
class DrawView: UIView {
    var currentLine: Line?
    var finishedLines = [Line]()
}
```

Demo – TouchTracker

- The view controller needs to know that its view will be an instance of **DrawView**
- Open Main.storyboard => Select the View and open the identity inspector => change the Class to DrawView



101

Demo – TouchTracker

- How to draw finished lines and current line (if any)?
- Solution: Use **UIBezierPath** to create and stroke a path

Demo – TouchTracker

- In DrawView.swift,

```
func stroke(_ line: Line) {
    let path = UIBezierPath()
    path.lineWidth = 10
    path.lineCapStyle = .round

    path.move(to: line.begin)
    path.addLine(to: line.end)
    path.stroke()
}
```


Demo – TouchTracker

- In DrawView.swift,

```
override func draw(_ rect: CGRect) {
    // Draw finished lines in black
    UIColor.black.setStroke()
    for line in finishedLines {
        stroke(line)
    }

    if let line = currentLine {
        // If there is a line currently being drawn, do it in red
        UIColor.red.setStroke()
        stroke(line)
    }
}
```

Demo – TouchTracker

- **Touches to Line**
  - In DrawView.swift, implement **touchesBegan(_:with:)** to start a new line.

```
override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
    let touch = touches.first!

    // Get location of the touch in view's coordinate system
    let location = touch.location(in: self)

    currentLine = Line(begin: location, end: location)

    setNeedsDisplay()
}
```


Demo – TouchTracker

- **Touches to Line**
  - In DrawView.swift, implement **touchesMoved(_:with:)** to update end of current line.

```
override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
    let touch = touches.first!
    let location = touch.location(in: self)

    currentLine?.end = location

    setNeedsDisplay()
}
```

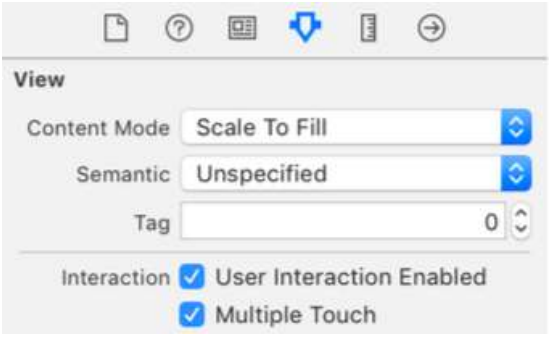
Demo – TouchTracker

- **Touches to Line**
  - In DrawView.swift, implement **touchesEnded(_:with:)** to update end of current line.

```
override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
    if var line = currentLine {
        let touch = touches.first!
        let location = touch.location(in: self)
        line.end = location

        finishedLines.append(line)
    }
    currentLine = nil

    setNeedsDisplay()
}
```


Demo – TouchTracker

- **Handling multiple touches**
  - In Main.storyboard, select the Draw View and open the attributes inspector => Check the box labeled Multiple Touch



Demo – TouchTracker

- **Handling multiple touches**
- replace the single **Line** with a dictionary containing instances of **Line**

```
class DrawView: UIView {

    var currentLine: Line?
    var currentLines = [NSValue:Line]()
```

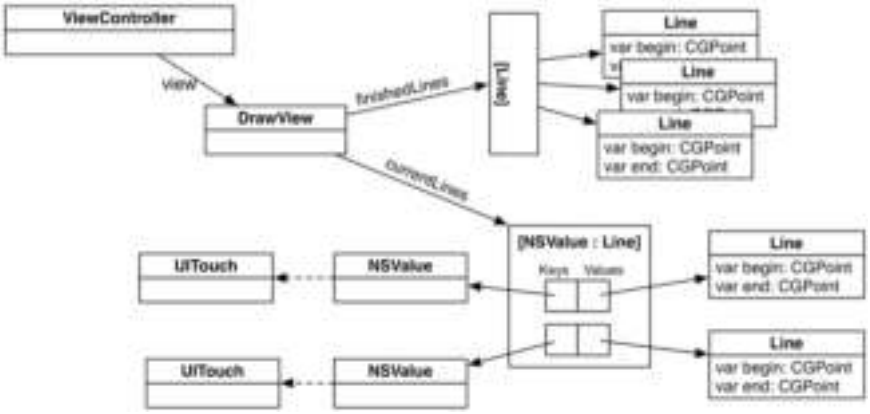

Demo – TouchTracker

- **Handling multiple touches**
  - In DrawView.swift, update the code in **touchesBegan(_:with:)**.

```
// Log statement to see the order of events
print(#function)

for touch in touches {
    let location = touch.location(in: self)

    let newLine = Line(begin: location, end: location)

    let key = NSValue(nonretainedObject: touch)
    currentLines[key] = newLine
}
```

Demo – TouchTracker

- **Handling multiple touches**
  - Object diagram for multitouch TouchTracker



Demo – TouchTracker

- **Handling multiple touches**
  - update **touchesMoved(_:with:)** in DrawView.swift

```
// Log statement to see the order of events
print(#function)

for touch in touches {
    let key = NSValue(nonretainedObject: touch)
    currentLines[key]?.end = touch.location(in: self)
}
```

Demo – TouchTracker

- **Handling multiple touches**
  - update **touchesEnded(_:with:)**

```
// Log statement to see the order of events
print(#function)

for touch in touches {
    let key = NSValue(nonretainedObject: touch)
    if var line = currentLines[key] {
        line.end = touch.location(in: self)

        finishedLines.append(line)
        currentLines.removeValue(forKey: key)
    }
}
```


Demo – TouchTracker

- **Handling multiple touches**
- update **draw(_:)** to draw each line in currentLines

```
// Draw current lines in red
UIColor.red.setStroke()
for (_, line) in currentLines {
    stroke(line)
}
```

Demo – TouchTracker

- A touch can be **canceled** when the application is interrupted by the OS (for example, when a phone call comes in) while a touch is currently on the screen.
- When a touch is canceled, any state it set up should be reverted.
  - In this case, we need to remove any lines in progress.

Demo – TouchTracker

- In DrawView.swift, implement **touchesCancelled(_:with:)**

```
override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?) {
    // Log statement to see the order of events
    print(#function)

    currentLines.removeAll()

    setNeedsDisplay()
}
```

Demo – TouchTracker

- Don’t want to set fix line colors in code
- Use **@IBInspectable**
  - When working in Interface Builder, you are able to modify attributes

Demo – TouchTracker

- **IBInspectable**

```
@IBInspectable var finishedLineColor: UIColor = UIColor.black {
    didSet {
        setNeedsDisplay()
    }
}

@IBInspectable var currentLineColor: UIColor = UIColor.red {
    didSet {
        setNeedsDisplay()
    }
}

@IBInspectable var lineThickness: CGFloat = 10 {
    didSet {
        setNeedsDisplay()
    }
}
```

Demo – TouchTrac

- **IBInspectable**

```
func stroke(_ line: Line) {
    let path = UIBezierPath()
    path.lineWidth = 10
    path.lineWidth = lineThickness
    path.lineCapStyle = .round

    path.move(to: line.begin)
    path.addLine(to: line.end)
    path.stroke()
}

override func draw(_ rect: CGRect) {
    // Draw finished lines in black
    UIColor.black.setStroke()
    finishedLineColor.setStroke()
    for line in finishedLines {
        stroke(line)
    }

    // Draw current lines in red
    UIColor.red.setStroke()
    currentLineColor.setStroke()
    for (_, line) in currentLines {
        stroke(line)
    }
}
```


Demo – TouchTracker

- **@IBInspectable**
- Customizing DrawView

GESTURE RECOGNITION

Gesture

- Detect a specific pattern of touches that make a gesture, like a pinch or a swipe

Touch Tracker - Gesture

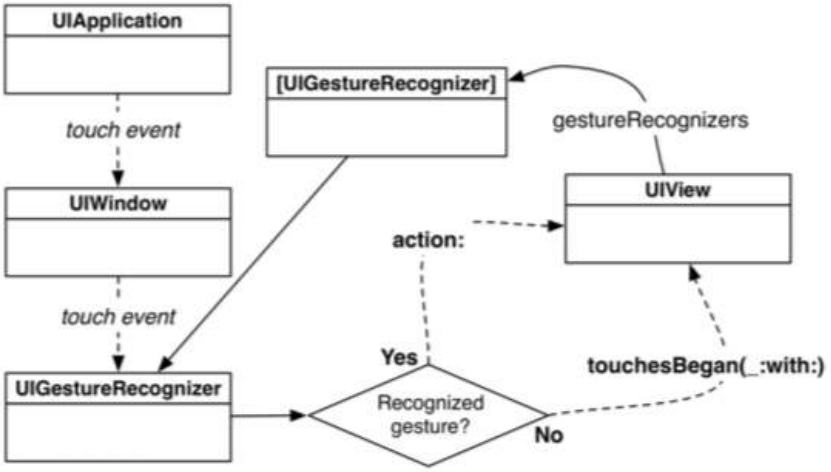
- Recognize tap and double tap

```
@objc func tap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a tap")
}

@objc func doubleTap(_ gestureRecognizer: UIGestureRecognizer) {
    print("Recognized a double tap")
    currentLines.removeAll()
    finishedLines.removeAll()
    setNeedsDisplay()
}
```


Touch Tracker - Gesture

```
required init?(coder aDecoder: NSCoder) {
    super.init(coder: aDecoder)
    let doubleTapRecognizer = UITapGestureRecognizer(target: self,
                                                     action: #selector(DrawView.doubleTap(_:)))
    doubleTapRecognizer.numberOfTapsRequired = 2
    doubleTapRecognizer.delaysTouchesBegan = true
    addGestureRecognizer(doubleTapRecognizer)
    let tapRecognizer =
        UITapGestureRecognizer(target: self, action: #selector(DrawView.tap(_:)))
    tapRecognizer.delaysTouchesBegan = true
    tapRecognizer.require(toFail: doubleTapRecognizer)
    addGestureRecognizer(tapRecognizer)
}
```

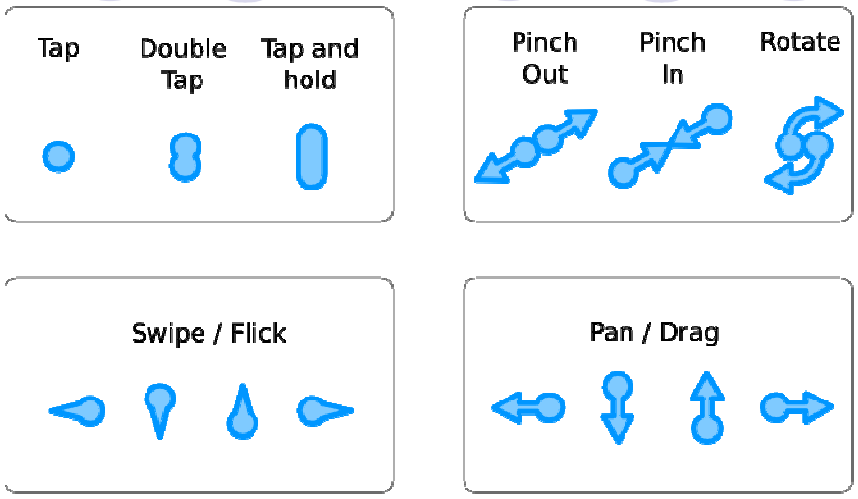
Gesture recognizers intercept touches



Gesture Recognizer

- A gesture recognizer is an object of the abstract class UIGestureRecognizer
- There are specific subclasses of it that are provided for usage, and each one of them deals with a specific kind of gesture.

Gesture Recognizer



Gesture Recognizer

- UITapGestureRecognizer:
  - It can be used to handle single or multiple taps, either with one or more fingers.
  - Tapping is one of the most usual gestures that users make.

Gesture Recognizer

- UISwipeGestureRecognizer:
  - Swiping happens when dragging a finger towards a direction (right, left, top and down).
  - A characteristic example of the swipe gesture exists on the Photos app, where we use our fingers to slide from one photo to another.

Gesture Recognizer

- UIPanGestureRecognizer:
  - The pan gesture is actually a drag gesture.
  - It's used when it's needed to drag views from one point to another.

Gesture Recognizer

- UIPinchGestureRecognizer:
  - When you view photos on the Photos app and you use your two fingers to zoom in or out to a photo, then you perform a pinch gesture.
  - An object of this class is usually handy to change the transform of a view, and more specifically its scale.
  - Using pinch gestures for example, you can implement zoom in and out to photos on your own apps.

Gesture Recognizer

- UIRotationGestureRecognizer:
  - In accordance to the previous gesture, rotation is used to rotate a view using two fingers.

Gesture Recognizer

- UILongPressGestureRecognizer:
  - An object of that class monitors for long press gestures happening on a view.
  - The pressing must last long enough in order to be detected, and the finger or fingers should not move a lot around the pressed point otherwise the gesture fails.

Gesture Recognizer

- UIScreenEdgePanGestureRecognizer:
  - This one is similar to the swipe gesture, but with a great difference: The finger movement should always begin from an edge of the screen.

GestureRecognizerDemo

- Create a new project
- Select Main.storyboard file and drag a Label object from the Object Library panel to the center of the view
- Click on Auto Layout => Resolve Auto Layout Issues menu and select the Reset to Suggested Constraints

GestureRecognizerDemo

- Open Assistant Editor panel
- Ctrl-click on the label and drag to a position just below the class declaration line in the Assistant Editor.
- Release the line and, in the resulting connection dialog, establish an outlet connection named **statusLabel**.

GestureRecognizerDemo

- add Tap, Pinch, Rotation, Swipe and Long Press Gesture Recognizer objects to the design

GestureRecognizerDemo

- Select the Tap Gesture Recognizer instance and display the Attributes Inspector.
- Within the attributes panel, change the Taps value to 2 so that only double taps are detected.
- Similarly, select the Long Press Recognizer object and change the Min Duration attribute to 3 seconds.

GestureRecognizerDemo

- Ctrl-click on the Tap Gesture Recognizer object and drag the line to the area immediately beneath the viewDidLoad method in the Assistant Editor panel.
- Name action “tapDetected”
- Similarly create actions: “pinchDetected”, “rotationDetected”, “swipeDetected”, “longPressDetected”

Implement action methods

- tapDetected

```
@IBAction func tapDetected(
    _ sender: UITapGestureRecognizer) {
    statusLabel.text = "Tap Detected..."
}
```

Implement action methods

- pinchDetected

```
@IBAction func pinchDetected(
    _ sender: UIPinchGestureRecognizer) {
    let scale = sender.scale
    let velocity = sender.velocity
    let resultString = "Pinch - \(scale), \(velocity)"
    statusLabel.text = resultString
}
```


Implement action methods

rotationDetected

```
@IBAction func rotationDetected(
    _ sender: UIRotationGestureRecognizer) {
    let radiance = sender.rotation
    let velocity = sender.velocity
    let resultString = "Rotation - \(radiance), \(velocity)"
    statusLabel.text = resultString
}
```

Implement action methods

- swipeDetected

```
@IBAction func swipeDetected(
    _ sender: UISwipeGestureRecognizer) {
    statusLabel.text = "Right Swipe Detected..."
}
```


Implement action methods

- longPressDetected

```
@IBAction func longPressDetected(
    _ sender: UILongPressGestureRecognizer) {
    statusLabel.text = "Long Press Detected..."
}
```

GestureRecognizerDemo

- Build and run the application.

UIMenuController

- The singleton *UIMenuController* instance is referred to as the editing menu.
- It is the menu interface for the Cut, Copy, Paste, Select, Select All, and Delete commands.

UIMenuController

- Select Main.storyboard file and drag a Button object from the Object Library panel to below the statusLabel
- Set Auto Layout

UIMenuController

- Ctrl+drag from button to ViewController.swift and create action “buttonTapped”

UIMenuController

- Create functions

```
@objc func clearLabel() {
    statusLabel.text = ""
}
override var canBecomeFirstResponder: Bool {
    return true
}
override func canPerformAction(_ action: Selector,
                               withSender sender: Any?) -> Bool {
    if action == #selector(ViewController.clearLabel) {
        return true
    }
    return false
}
```


UIMenuController

- buttonTapped()

```
@IBAction func buttonTapped(_ sender: UIButton) {
    becomeFirstResponder()
    let menu = UIMenuController.shared
    let clearItem = UIMenuItem(title: "Clear Text",
                               action:
        #selector(ViewController.clearLabel))
    menu.menuItems = [clearItem]
    menu.setTargetRect(CGRect(x: 0, y: 0, width: 20, height: 20),
                       in: statusLabel)
    menu.setMenuVisible(true, animated: true)
}
```

GestureRecognizerDemo

- Build and run the application.

Questions ???