iOS

Some first elements
Development Environment

- Platforms
  - iOS
  - WatchOS
  - TvOS

- Tools
  - Xcode 8.2 (emulator)

- Language
  - Swift 3.0.1

- Frameworks
  - MVC
Four layers

- The Cocoa Touch layer contains key frameworks for building iOS apps
  - These frameworks define the appearance of your app
  - They also provide the basic app infrastructure and support for key technologies such as multitasking, touch-based input, push notifications, and many high-level system services
- The Media layer contains the graphics, audio, and video technologies you use to implement multimedia experiences in your apps
- The Core Services layer contains fundamental system services for apps
  - This layer also contains individual technologies to support features such as location, iCloud, social media, and networking
- The Core OS layer contains the low-level features that most other technologies are built upon
ApplicationDelegate

- A single window, where all of our app content is drawn
- “Skeletons” of important methods that allow the application object to talk to the app delegate
  - During runtime events (e.g., app launch, low-memory warnings, and app termination) the application object calls the corresponding method in the app delegate, giving it an opportunity to respond appropriately
- Delegate design pattern
import UIKit

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions launchOptions: [NSObject: AnyObject]?) -> Bool {
        // Override point for customization after application launch.
        return true
    }

    func applicationWillResignActive(application: UIApplication) {
        // Sent when the application is about to move from active
        // to inactive state
    }

    func applicationDidEnterBackground(application: UIApplication) {
        // Use this method to release shared resources, save user data,
        // invalidate timers, and store application state
    }

    func applicationWillEnterForeground(application: UIApplication) {
        // Called as part of the transition from the background to the inactive
        // state
    }

    func applicationDidBecomeActive(application: UIApplication) {
        // Restart any tasks that were paused while the application was inactive
    }

    func applicationWillTerminate(application: UIApplication) {
        // Called when the application is about to terminate
    }
}
Views

• Represent user interface elements that display contents or respond to user events
  — Can be nested in a view hierarchy
  — Can animate their property values

• Views do not know the role they play
  — For example, clicking a button is always the same, it does not know what it controls
View Controllers

• Provide the infrastructure for managing content and for coordinating the showing and hiding of it
  — Manage the views used to display content
  — Communicate and coordinate with other view controllers when transitions occur
• Different view controllers can control separate portions of your user interface
• May also communicate with other controllers, such as data controllers or document objects
View Controllers

• You use custom subclasses of UIViewController to present your app’s content
  – Content view controllers
    • UIViewController, UITableViewController, UICollectionViewController
  – Container view controllers
    • UINavigationController, UITabBarController, UISplitViewController
Views and view Controllers

• Every view is controlled by only one view controller
  — When a view is assigned to the view controller’s view property, the view controller owns it
• If the view is a subview, it might be controlled by the same view controller or a different view controller
Graphical elements (1)
View Controller Hierarchy (II)
Overall organization (storyboard)
Storyboards

• A Storyboard is a visual representation of the app’s user interface
  – Shows screens of content and the transitions between them
• A Scene Corresponds to a Single View Controller and Its Views
  – On iPhone, each scene corresponds to a full screen’s worth of content
  – On iPad, multiple scenes can appear on screen at once
  – Scenes are linked together with Segues, these define the transitions between one scene to another
Unified Storyboards for Universal Apps

• Create a single interface for your app that works well on both iPad and iPhone, adjusting to orientation changes and different screen sizes as needed

• Design apps with a common interface and then customize them for different size classes

• You no longer need to create a specific iPad storyboard; instead target the appropriate size classes and tune your interface for the best experience
How to start

Choose a template for your new project:

**Application**
- Single View Application
- Game
- Master-Detail Application
- Page-Based Application
- Tabbed Application
- Sticker Pack Application
- iMessage Application

**Framework & Library**
- Cocoa Touch Framework
- Cocoa Touch Static Library
- Metal Library
Two choices

Choose options for your new project:

Product Name: 
Team: Add account...
Organization Name: Luciano Baresi
Organization Identifier: it.polimi
Bundle Identifier: it.polimi.ProductName
Language: Swift
Devices: Universal

- Use Core Data
- Include Unit Tests
- Include UI Tests

Cancel Previous Next
### General

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Name</td>
<td>newApp</td>
</tr>
<tr>
<td>Bundle Identifier</td>
<td>it.polimi.newApp</td>
</tr>
<tr>
<td>Version</td>
<td>1.0</td>
</tr>
<tr>
<td>Build</td>
<td>1</td>
</tr>
</tbody>
</table>

### Signing

No accounts found. Add a developer account to sign your app.

[Add Account...]

### Deployment Info

- **Deployment Target**: **iOS 10.2**
- **Devices**: Universal
- **Main Interface**: Main
- **Device Orientation**: Portrait
  - Checked: Portrait, Landscape Left, Landscape Right
- **Status Bar Style**: Default
  - Checked: Hide status bar, Requires full screen

### App Icons and Launch Images

- **App Icons Source**: AppIcon
- **Launch Images Source**: Use Asset Catalog
- **Launch Screen File**: LaunchScreen

### Embedded Binaries

Add embedded binaries here
iOS Simulator

- Easy to start and try apps
- Known problems
  - Different devices
  - Different orientations
- Better positioning of elements
How to add graphical elements

Do you want to send a message?

- Send

Button

Implements a button that intercepts touch events and sends an action message to a target object when it’s tapped. You can set the title, image, and other appearance properties of a button. In addition, you can specify a different appearance for each button state.
Views (I)

Action sheet

Activity indicator

Alert view

Collection view

Image view

The volume of the ringer and alerts can be adjusted using the volume buttons.

Label

Navigation bar and items
Views (II)

Downloading 30 of 108

Progress view

Search bar

Tab bar

Table view

Sent from my iPhone

Text view

Tool bar

Web view
Properties

- **UIColor**
  - UIColor.green

- **UIFont**
  - UIFont.preferredFontForTextStyle()
  - You should never use fonts directly
UILabel

- Implements a read-only text view
- Set label content using fields text or attributedText
- By default, a label is a single line
  - To create a multiline label, we must increase the value of field numberOfLines
- Special-purpose fields can be used for setting the look and feel of all characters
UITextView

- Implements the behavior of a scrollable, multiline text region
- Supports the display of text using custom style information and also supports text editing
- The appearance of the keyboard itself can be customized using the properties provided by protocol UITextInputTraits
Autolayout

- Determines where objects should go and how big they should be based on constraints we set on them
  - This allows interfaces to adapt to being rotated between portrait and landscape, and to handle differing screen sizes
- Constraints allow us to express what matters to us and to let other factors vary as needed
  - We can specify the size of components, their alignment with or distance from other components, etc.
<table>
<thead>
<tr>
<th>Auto Layout Attributes</th>
<th>Value</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>The size of the view.</td>
<td>These attributes can be assigned constant values or combined with other Height and Width attributes. These values cannot be negative.</td>
</tr>
<tr>
<td>Width</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Top</td>
<td>The values increase as you move down the screen.</td>
<td>These attributes can be combined only with Center Y, Top, Bottom, and Baseline attributes.</td>
</tr>
<tr>
<td>Bottom</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Baseline</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Leading</td>
<td>The values increase as you move towards the trailing edge. For a left-to-right layout directions, the values increase as you move to the right. For a right-to-left layout direction, the values increase as you move left.</td>
<td>These attributes can be combined only with Leading, Trailing, or Center X attributes.</td>
</tr>
<tr>
<td>Trailing</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left</td>
<td>The values increase as you move to the right.</td>
<td>These attributes can be combined only with Left, Right, and Center X attributes. Avoid using Left and Right attributes. Use Leading and Trailing instead. This allows the layout to adapt to the view's reading direction. By default the reading direction is determined based on the current language set by the user. However, you can override this where necessary. In iOS, set the <code>semanticContentAttribute</code> properly on the view holding the constraint (the nearest common ancestor of all views affected by the constraint) to specify whether the content's layout should be flipped when switching between left-to-right and right-to-left languages.</td>
</tr>
<tr>
<td>Right</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Center X</td>
<td>The interpretation is based on the other attribute in the equation.</td>
<td>Center X can be combined with Center X, Leading, Trailing, Right, and Left attributes. Center Y can be combined with Center Y, Top, Bottom, and Baseline attributes.</td>
</tr>
<tr>
<td>Center Y</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Four buttons

• Stack layouts views in either a column or a row. It applies Auto Layout to the views without you having to add the constraints.

• Align creates constraints that align a view’s edges or horizontal or vertical center with another (containing) view

• Pin creates constraints that specify a fixed value for spacing views and/or define a fixed width or height

• Resolve adjusts a view position or size so it matches its constraints, or do the opposite and create constraints based on its current position and size
Example

Do you want to send a message?

Send
Constraints and colors

- Lines with T-shaped end-caps show the size of a space
- Plain lines show where edges align
- Solid Lines represent required constraints
- Dashed Lines represent optional constraints
- Red Lines say that one of the items affected by this constraint has an error
- Orange Lines indicate that the frame of one of the items affected by this constraint is not in the correct position
- Blue Lines say that the items affected by the constraint have a nonambiguous, satisfiable layout
App content

• Xcode provides a library of objects
  — Some of these are user interface elements that belong to a view, such as buttons and text fields
  — Others define the behavior of our app, such as view controllers and gesture recognizers
• A view controller manages a corresponding view and its sub-views
Interface Builder and code

- An IBOutlet connects a variable or property in code to an object in a storyboard
  - This lets us read and write objects’ properties, like reading the value of a slider or setting the initial contents of a text field
- An IBAction connects an event generated by a storyboard object to a method in the code
  - This lets us respond to a button being tapped or a slider’s value changing
Generated code

```swift
// ViewController.swift
// TestApp
// Created by Luciano Baresi on 1/5/15.// Copyright (c) 2015 Luciano Baresi. All rights reserved.

import UIKit

class ViewController: UIViewController {
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }
}
```
Control-click, drag
... and the result is
Logging

• On iOS, we can use function print() to write a string out to the system’s log file
  – For example, we can implement our action to just log a message every time the button is tapped

```swift
@IBAction func sendMessage(_ sender: UIButton) {
    print ("Button pressed")
}
```
A first complete example

```swift
@IBAction func sendMessage(_ sender: UIButton) {
    let alert = UIAlertController(title: "Sent",
                                   message: "First App Done",
                                   preferredStyle: .alert)
    alert.addAction(UIAlertAction(title: "Continue",
                                 style: .default, handler: nil))
    self.present(alert, animated: true, completion: nil)
}
```
Outlets

• Preceding a property with the `@IBOutlet` modifier tells Interface Builder that a property can serve as an outlet.

• A stored property implies a variable to store the value, but it is managed by Swift and we cannot access it directly.
  — Swift sets up getter and setter methods for us.

• A computed property does not imply a backing variable, but instead has `get()` and `set()` methods that we provide to compute the property’s value on the fly.
import UIKit

class ViewController: UIViewController {

    // ViewDidLoad
    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after
        // loading the view, typically from
        // a nib.

        super.viewDidLoad()
        // Dispose of any resources that can
        // be recreated.

        override func didReceiveMemoryWarning() {
            super.didReceiveMemoryWarning()
        }

        @IBAction func increment(sender: UIButton) {
        }
    }
}

// ViewController.swift
// TestApp
//
// Created by Luciano Baresi on 1/6/15.
// Copyright (c) 2015 Luciano Baresi. All
// rights reserved.
import UIKit

class ViewController: UIViewController {

    @IBOutlet weak var counterLabel: UILabel!

    var counter = 0

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading
        // the view, typically from a nib.
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that
        // can be recreated.
    }

    @IBAction func increment(_ sender: UIButton) {
        counter += 1
        counterLabel.text = "\(counter)"
    }
}
Weak attribute

- Automatic Reference Counting (ARC) solves almost all memory problems, but it cannot solve retain cycles
- Our ViewController knows about the UILabel, so ARC cannot free the label from memory as long as the view controller exists
- But if the UILabel also requires the ViewController, then neither can ever be freed from memory (cycle)
- The way to break this is to declare one side of the arrangement as weak, meaning that we do not require an object to be in memory if ours is the only one that knows about it
Why it works

• The top-level view has strong references to all its children (including the label)
• The view controller has a strong reference to the view, but an additional strong reference from the view controller to the label would be overkill
• The rule of thumb is that
  — Only “top-level” objects in a storyboard scene (like the view) need strong references, and everything else can be weak
  — Xcode defaults to this behavior when we made the connection