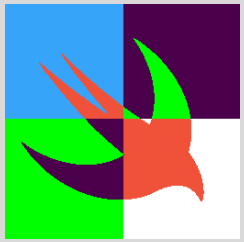
Mobile App Development - iOS

CSCI C323 – Summer 2020

Homework 01

due at 12:45PM on Monday, 2020-05-18

Instructor:
Mitja Hmeljak
http://mitja.info
mitja@indiana.edu



Homework 01: a first "FlashCards" app (i.e. "FlashCards" v.01)

What needs to be completed for Homework 01:

- follow instructions on ***all*** these slides
- make sure that your "*FlashCards*" *app v.01* works as expected
- turn in your Xcode project folder,
  to your personal IU GitHub repository for C323/Summer 2020,
  under **/hw/hw01/**
- turn in your personal README file, also under **/hw/hw01/**

- **If** you missed Lab02,
  please ***first*** follow all Lab 02 notes to create your personal IU GitHub repository for C323/Summer 2020, as from:
  https://homes.luddy.indiana.edu/classes/summer2020/csci/c323-mitja/2020/labs/lab02-c323-Summer-2020-IU-github.html
  then complete this Homework 01.

Homework 01: a first "FlashCards" app (i.e. "FlashCards" v.01)

in Homework 01,
we'll start writing an iOS app that does *two things*:

1. has buttons (two of them!)
2. responds to buttons (both of them!)

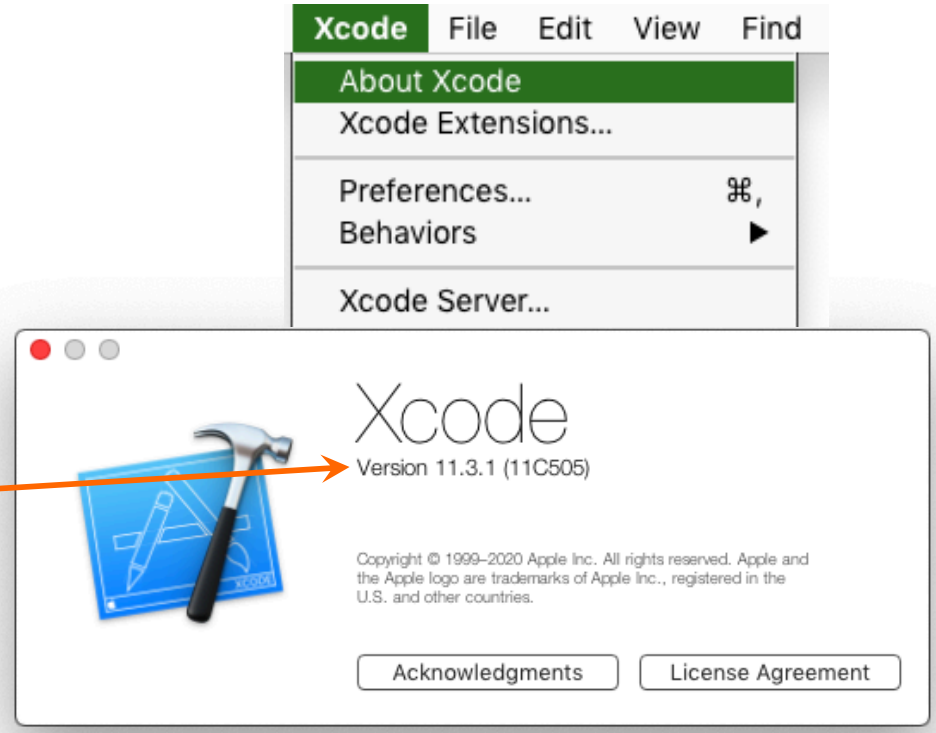
for this Homework 01,
we'll prepare the View and Controller parts of the app.

(the Model part will be part of a subsequent homework)

quick check: are you running Xcode 11.3.1 on your Mac?

Start *Xcode 11.3.1*,
and select the menu
Xcode →
About Xcode

Verify that you're running Xcode 11.3.1



If not, follow instructions at:
https://homes.luddy.indiana.edu/classes/summer2020/csci/c323-mitja/2020/resources/install-Xcode-11.html
to **install Xcode 11.3.1** on your Mac.

We can't accept any submission completed in *different Xcode versions*:
all such submissions will have to be rejected and given 0 points.

Writing your iOS App: "FlashCards"

Start *Xcode 11.3.1*,

and select the menu

File →

New →

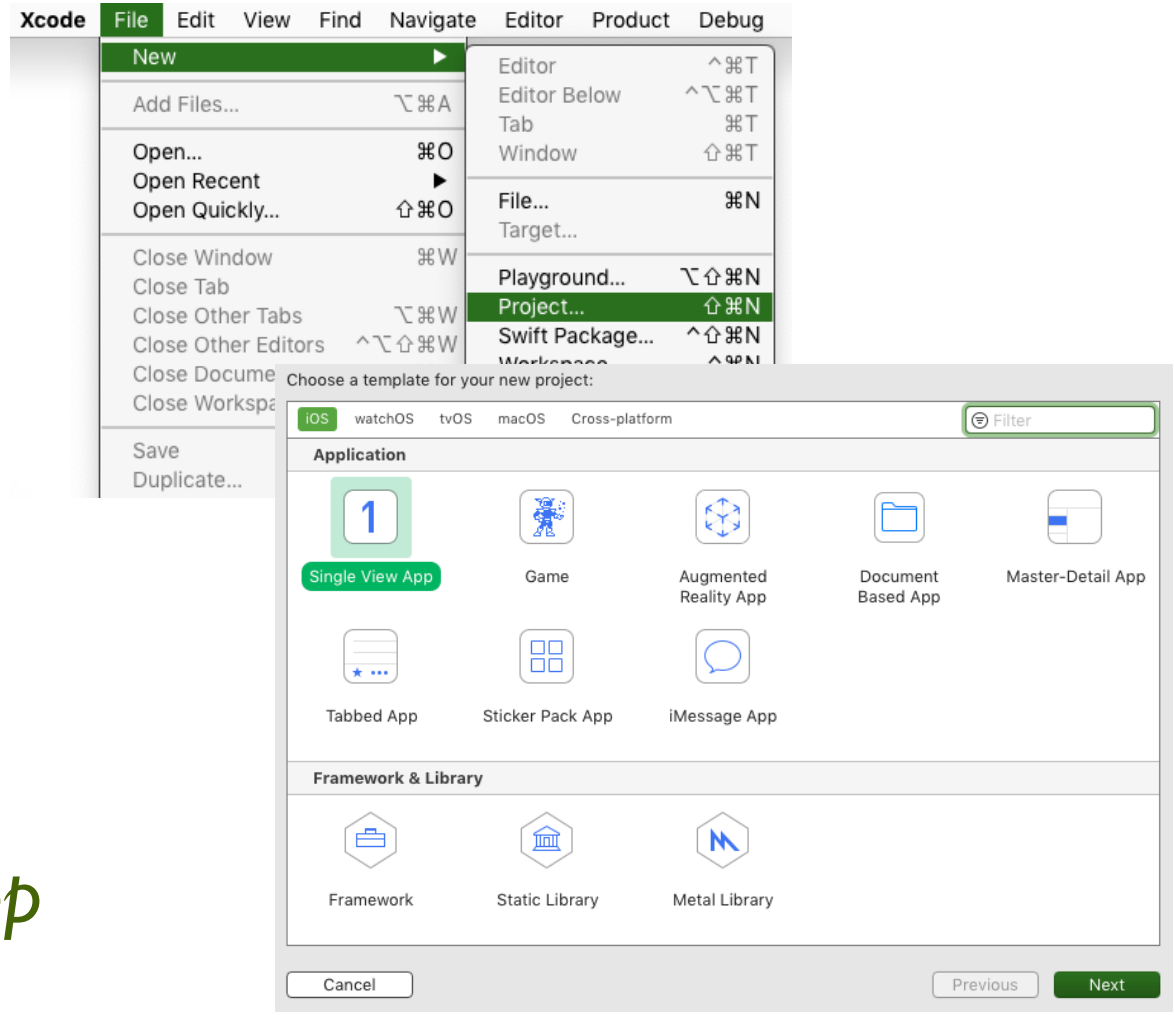
Project... :

Select

iOS →

Application →

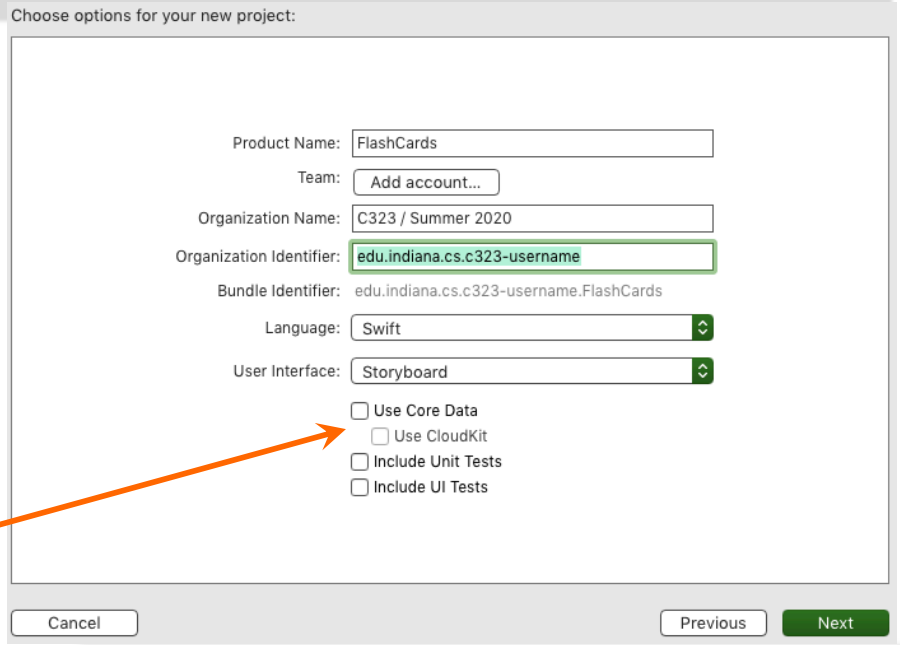
Single View App

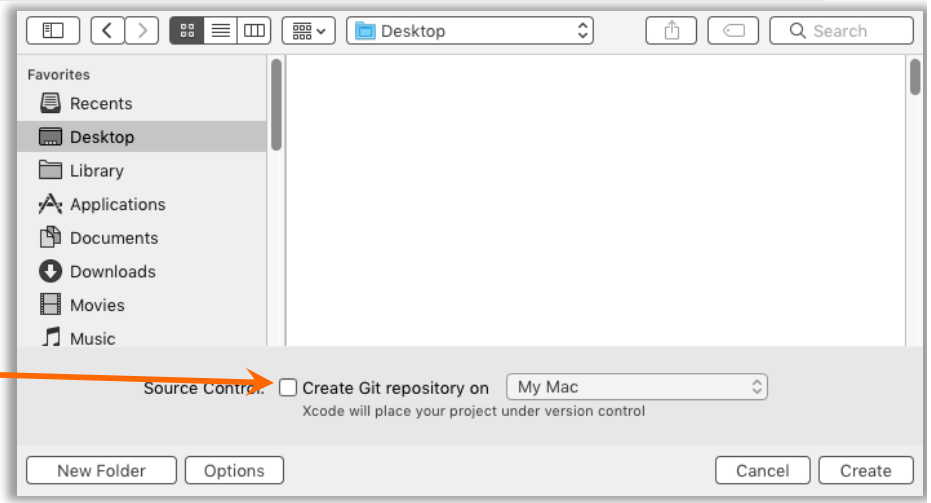


...then click *Next*...

Writing your iOS App: "FlashCards"

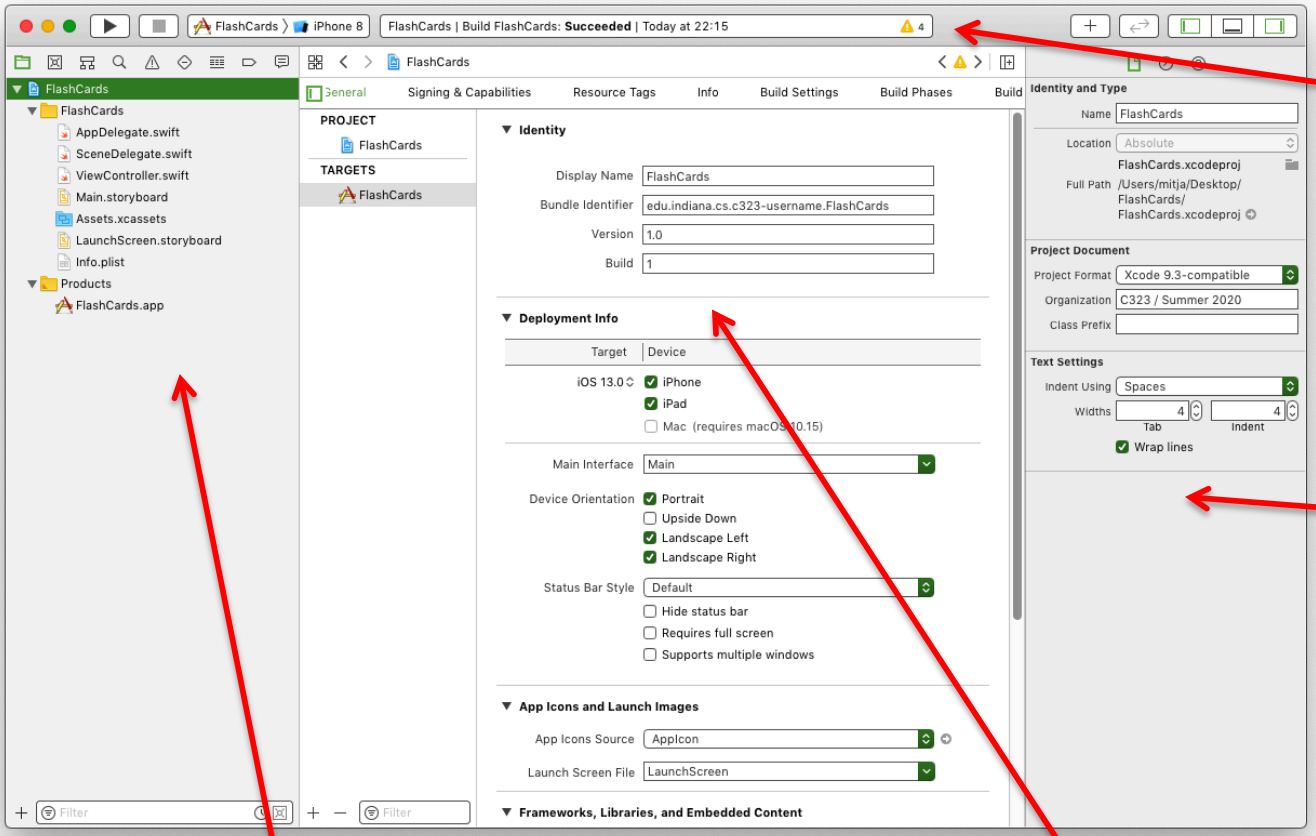
- name the app "**FlashCards**",
- (you don't need to choose a team, but you may do so, if you want to use a Developer account),
- set *your own first and last name* as "Organization Name"
- set the organization identifier to: `edu.indiana.cs.c323-`*username* (where *username* is your IU username),
- set *Language* to *Swift*,
- set User Interface to *Storyboard*, do *not* select "~~use Core Data~~", etc. (as shown here)
- then click *Next* ...
  then save the project to the Mac's *Desktop*, i.e.:
- ... select "Desktop" either from the sidebar or from the filesystem navigation pop-up menu,
- **deselect** "~~Source Control~~",
- and click *Create*.





Getting around Xcode 11.3.1

the Xcode workspace window has several parts:



Toolbar at the top of the window

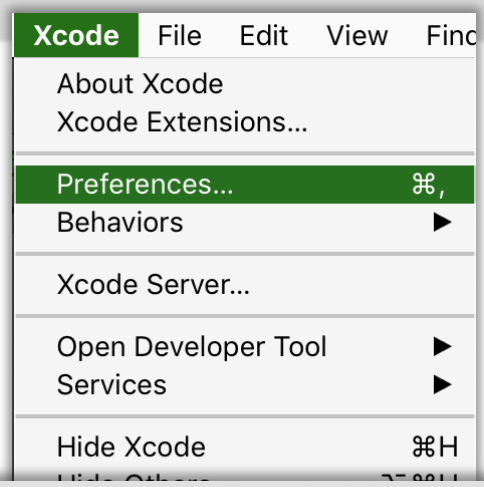
Utility area (inspectors) on the right side

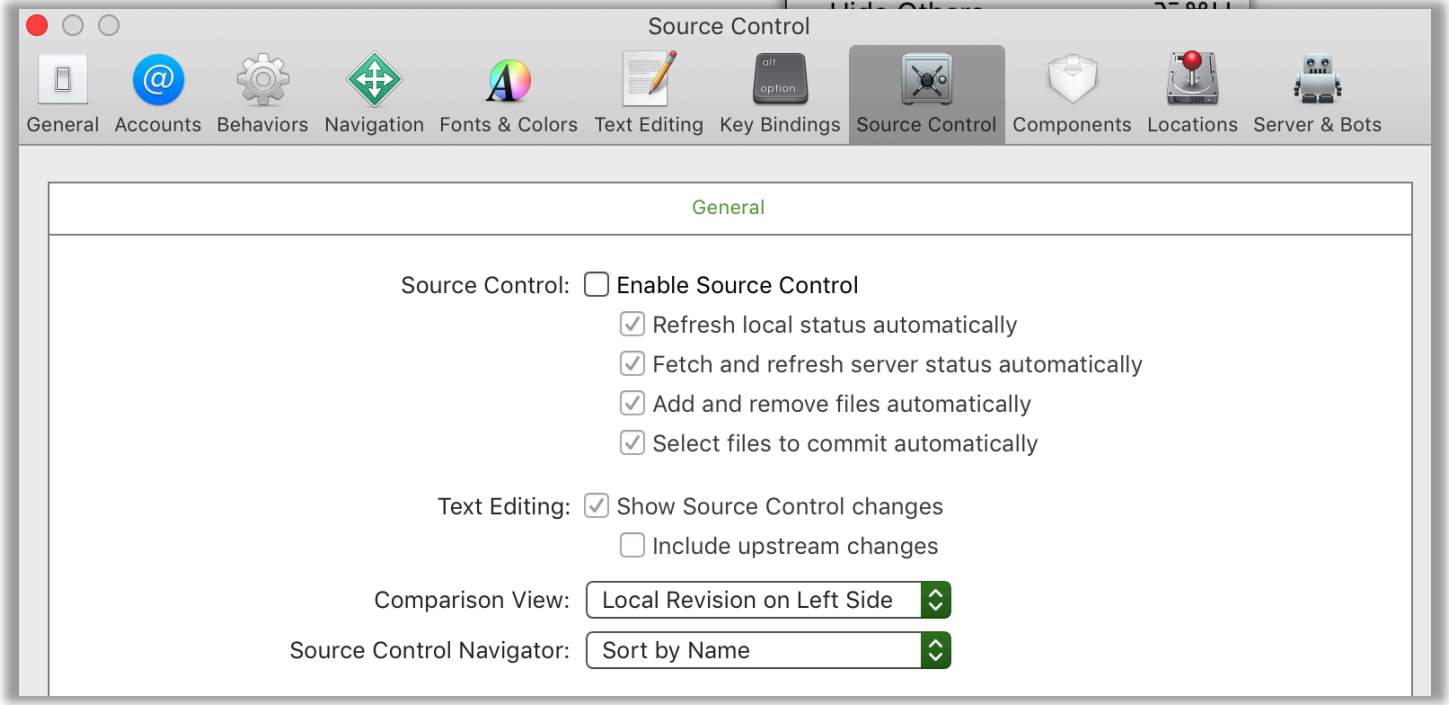
Navigator area (navigators) on the left side

Editor area in the center

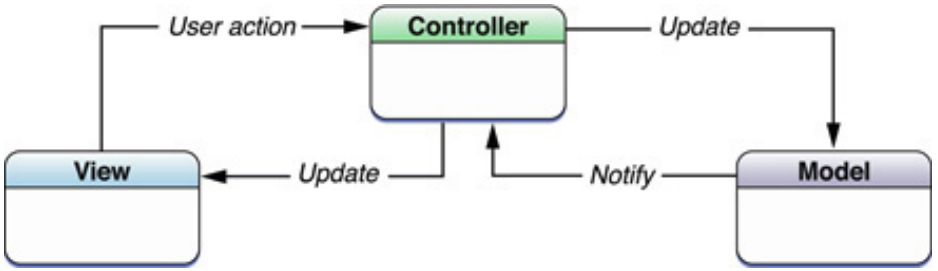
An iOS App: "FlashCards"

- while in Xcode, select the menu:
  - *Xcode → Preferences...*
- in the *Preferences* window, select *Source Control*
- in the *Source Control* pane,
  deselect "Enable Source Control":





the M-V-C design for the "FlashCards" app



View Objects: we'll need

- two UILabel instances
- two UIButton instances
- (four view objects total)

Model Objects: we'll need

- two *Array* object instances
  we'll use the Array type, which is a "collection type" in Swift
  - (note → we are *not* going to build this part for Homework 01, but in a *subsequent homework*)

Controller Objects: we'll need

- one AppDelegate object instance
- one ViewController object instance
  - (note → we are only partially writing this part now – we are going to make it more functional in a *subsequent homework*)

how do we plan and design an M-V-C application?

- *before* we start coding our M-V-C application,

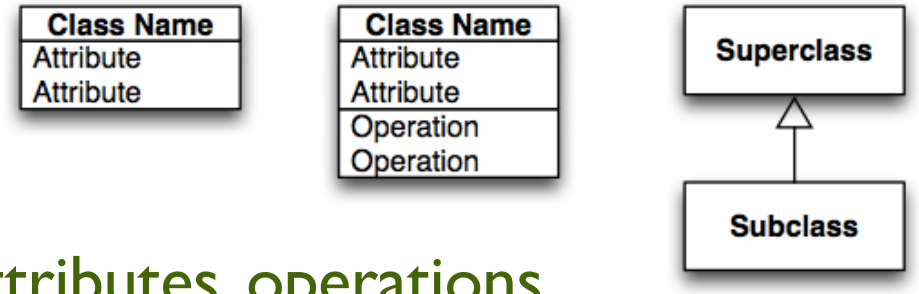
- we want to be able to *design* the application using the M-V-C pattern
  (please review the M-V-C design pattern in lecture notes)

- for our design needs, we'll use a *standardized design tool* called **U M L** :

- the → **Unified Modeling Language** ←

UML, the Unified Modeling Language, will help us with the M-V-C design of our app

(a 3-letter acronym: ***UML****)* the **U**nified **M**odeling **L**anguage, defines *many* possible ways to visualize a software system as a *diagram*, offering *views* of a *system model*.



In particular, UML defines:

- ***structure*** diagrams
  - static structure: objects, attributes, operations, relationships
- ***behavior*** diagrams: use cases, interaction diagrams such as sequences and timings, activity diagrams, state machines

UML diagrams help understanding connections between software objects.

We've used UML diagrams e.g. in Lecture 02, when describing the M-V-C design for the *FlashCards* app.

the M-V-C design for the "FlashCards" app

for the M-V-C design of our application,

we use an UML diagram

to map and categorize all objects into three blocks:

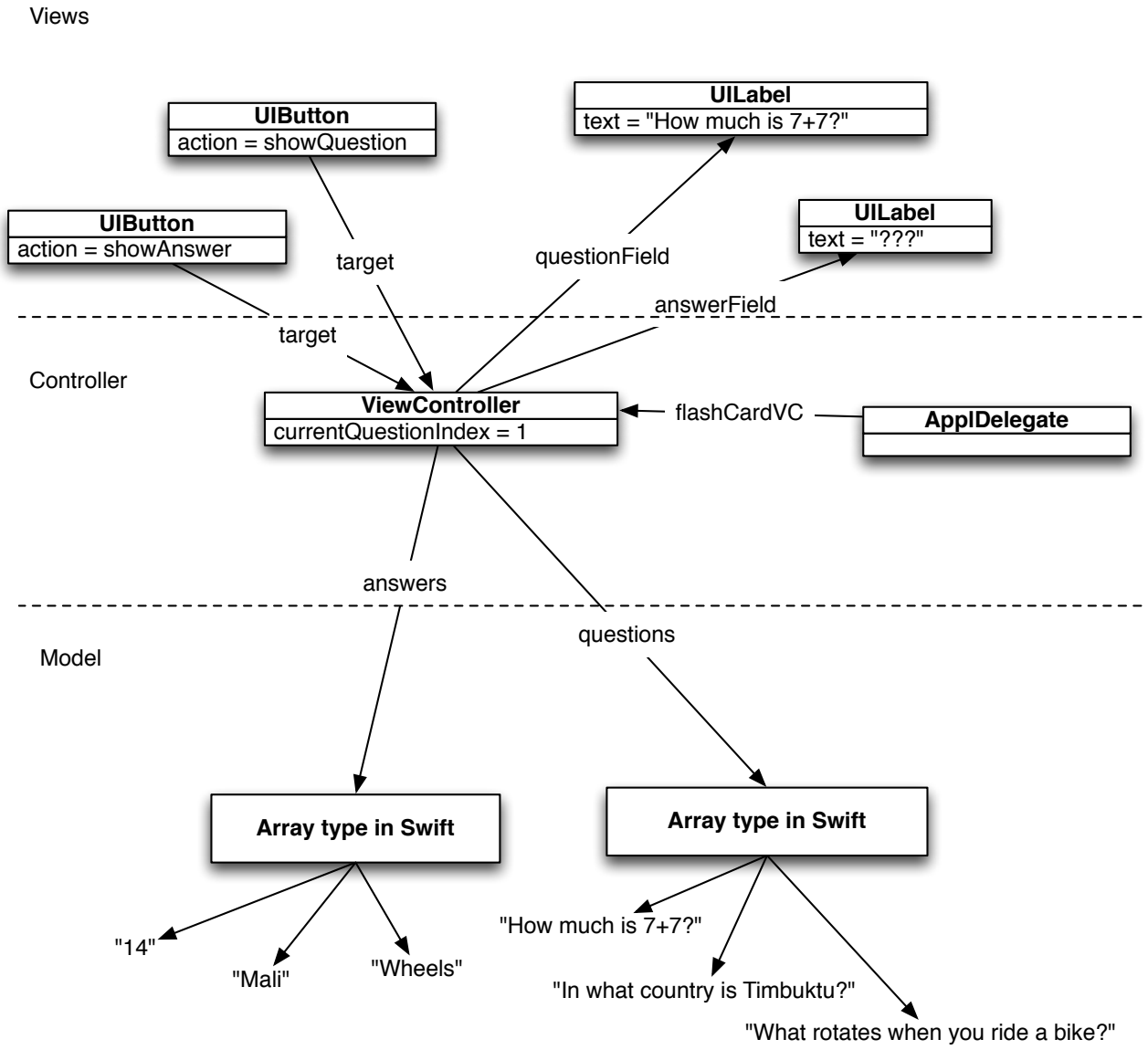
- *View Objects*
- *Controller Objects*
- *Model Objects*

the M-V-C design for the "FlashCards" app

View Objects:

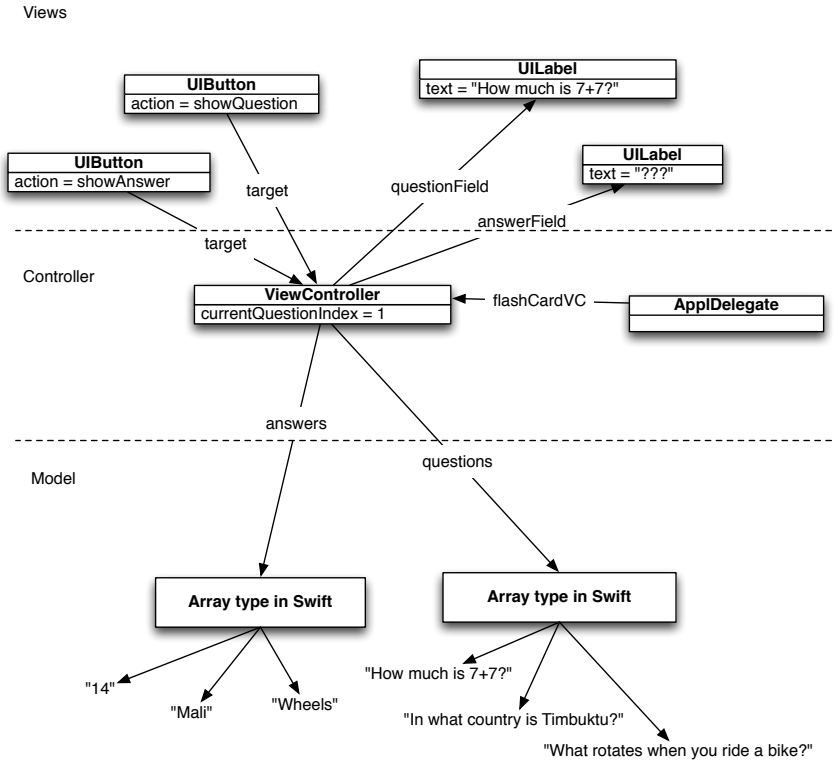
Controller Objects:

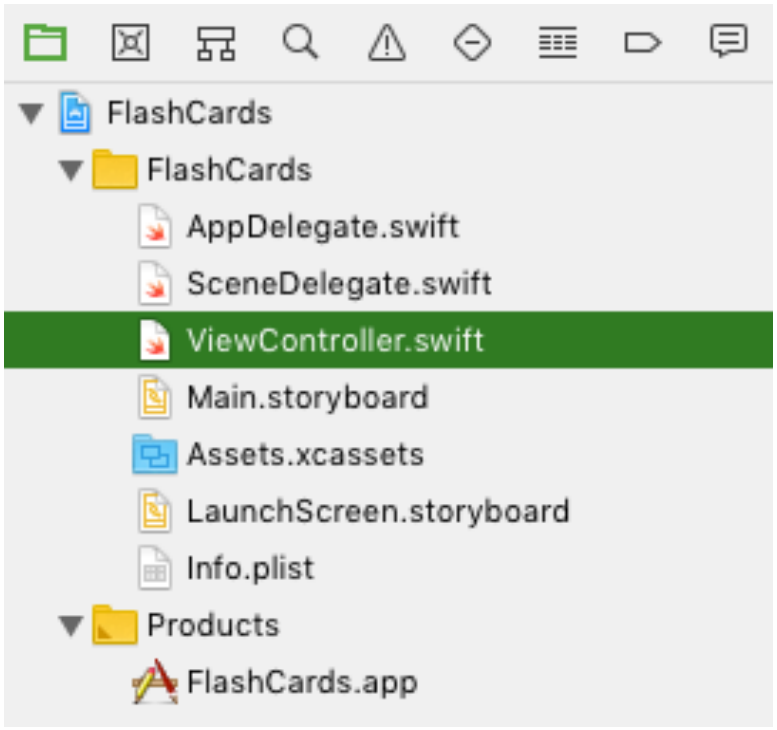
Model Objects:



the M-V-C design for the "FlashCards" app

with this overview of the objects we're going to use in our application, we can start building them in our Xcode project...

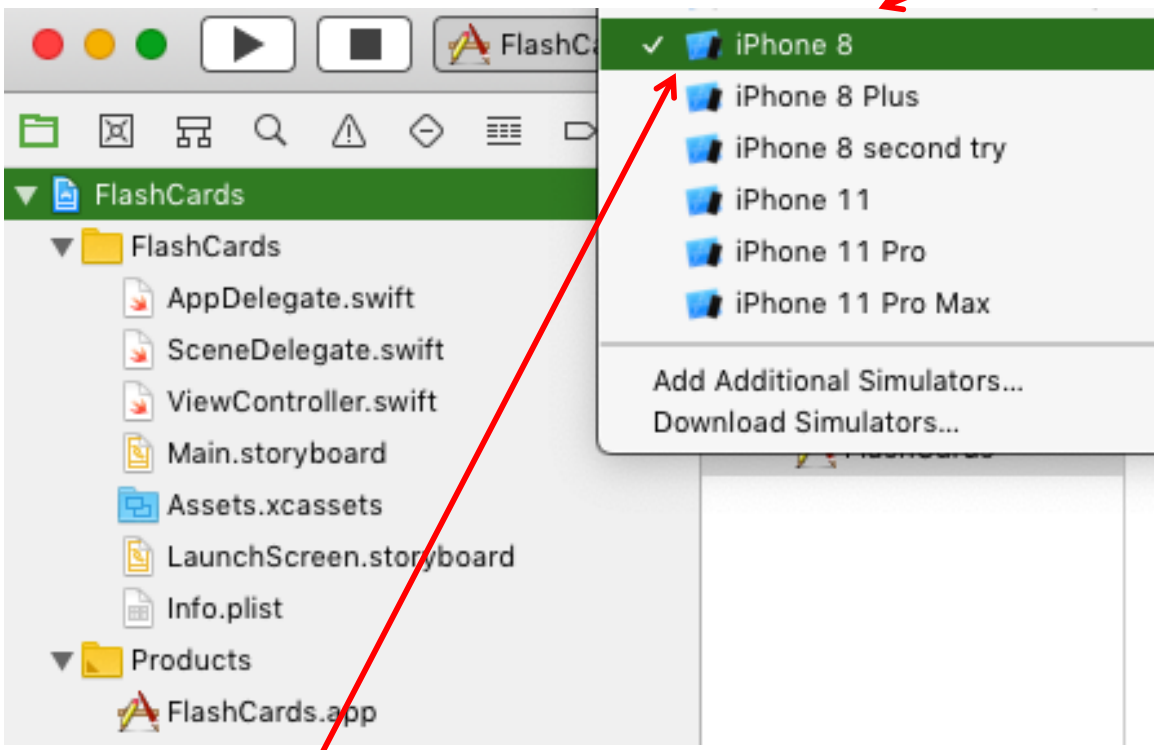




note: how are going to test *running* our "FlashCards" app?

we're going to test running our "***FlashCards***" app with the **iOS Simulator** as provided with Xcode.

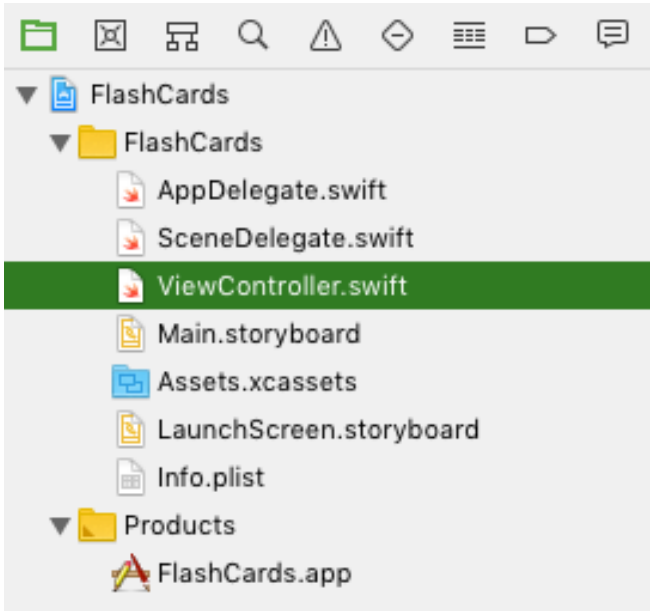
In your "***FlashCards***" project, **select** the *iPhone 8 simulator*



from the pop-up menu next to the "***FlashCards***" project name.

the View Controller for the "FlashCards" app

In your *Xcode* project, you'll find *these* files in the *project navigator* :
(among other files as well)



- a Swift *source code* file named "ViewController.swift"
- and a *Storyboard* type of file named "Main.storyboard"

In the filesystem, these two files are located thus:

- the *.swift* file is stored inside the "***FlashCards***" *source code folder* (which is inside the "***FlashCards***" *project folder*)
- and the *.storyboard* file is inside a folder named "Base.lproj" stored inside the *source code folder*.

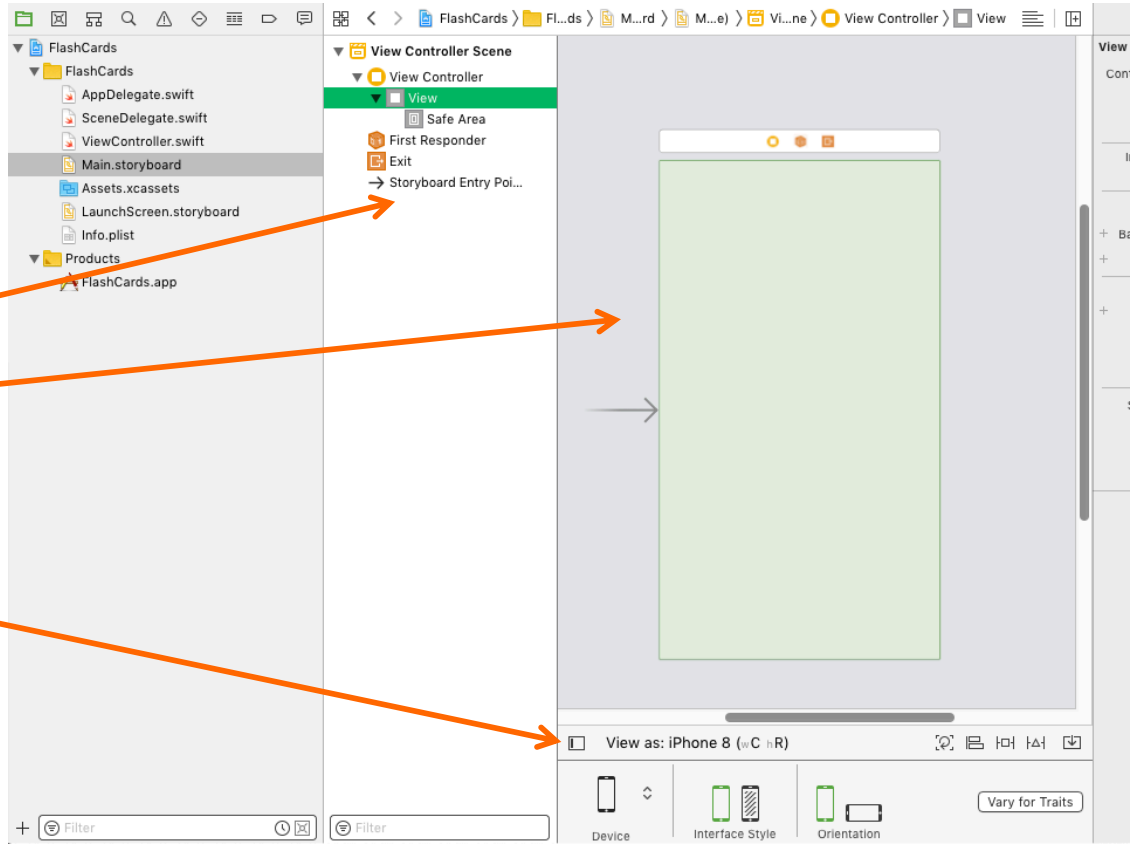
the View Controller for the "FlashCards" app

if you select the *Main.storyboard* file in the *project navigator*, it will open inside the main editor area, i.e. the ***Interface Builder*** tool in Xcode, where you can add, edit, arrange GUI objects:

There are two parts in the *editor* area in the center of the Xcode window:

- the editor's *dock* on the left
- the *canvas* on the right

(*to reveal the dock*, you may have to *click* on the "disclosure button" on the bottom left of the canvas)



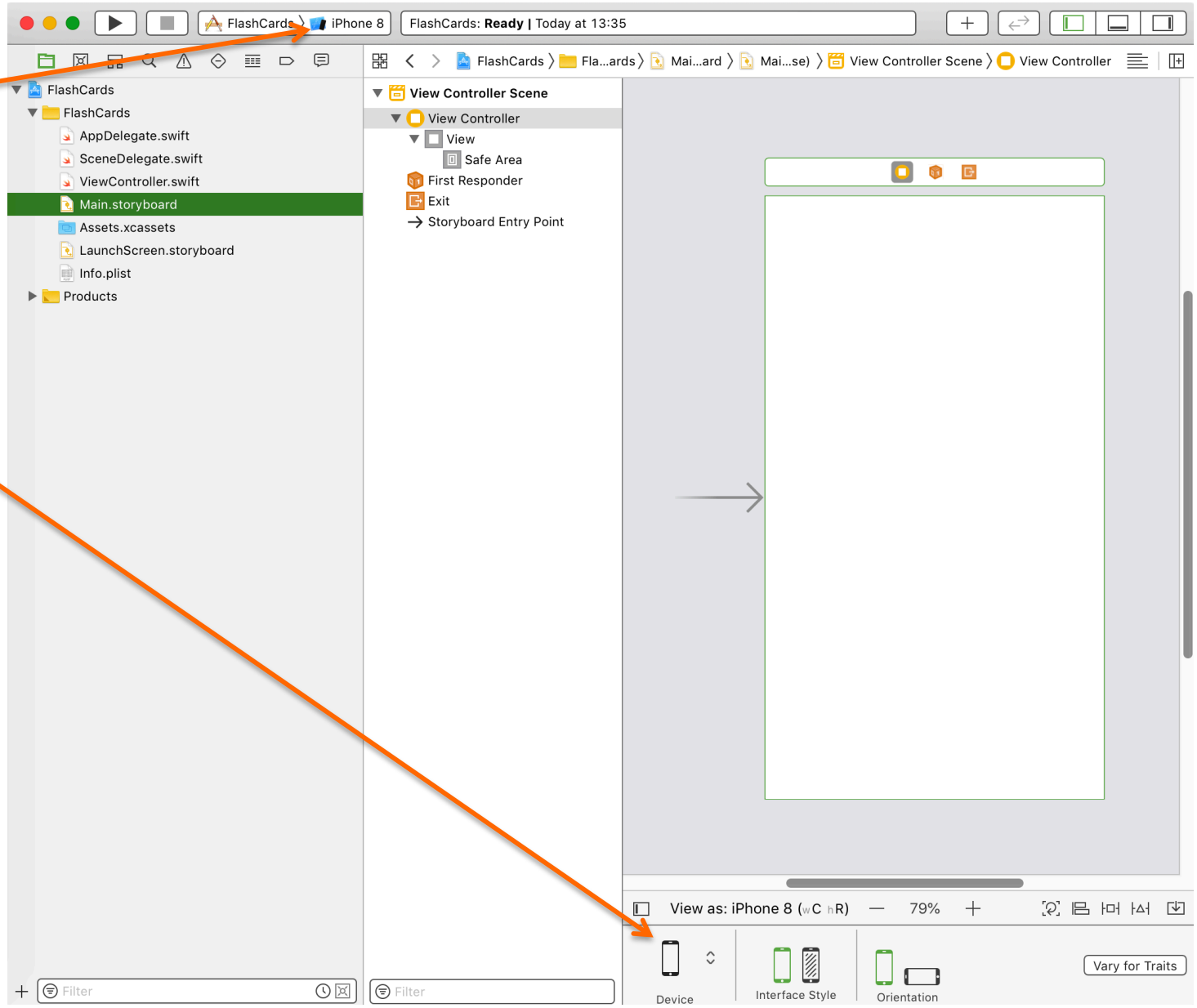
configure Xcode and Interface Builder for the "FlashCards" app

in the Xcode toolbar,

- **select iPhone 8**
  as target device
  for running the app
  in the iOS
  Simulator

in Interface Builder,

- **select iPhone 8**
  as displayed device
  for editing the
  Main.storyboard
  for the app.



building the View for the "FlashCards" app

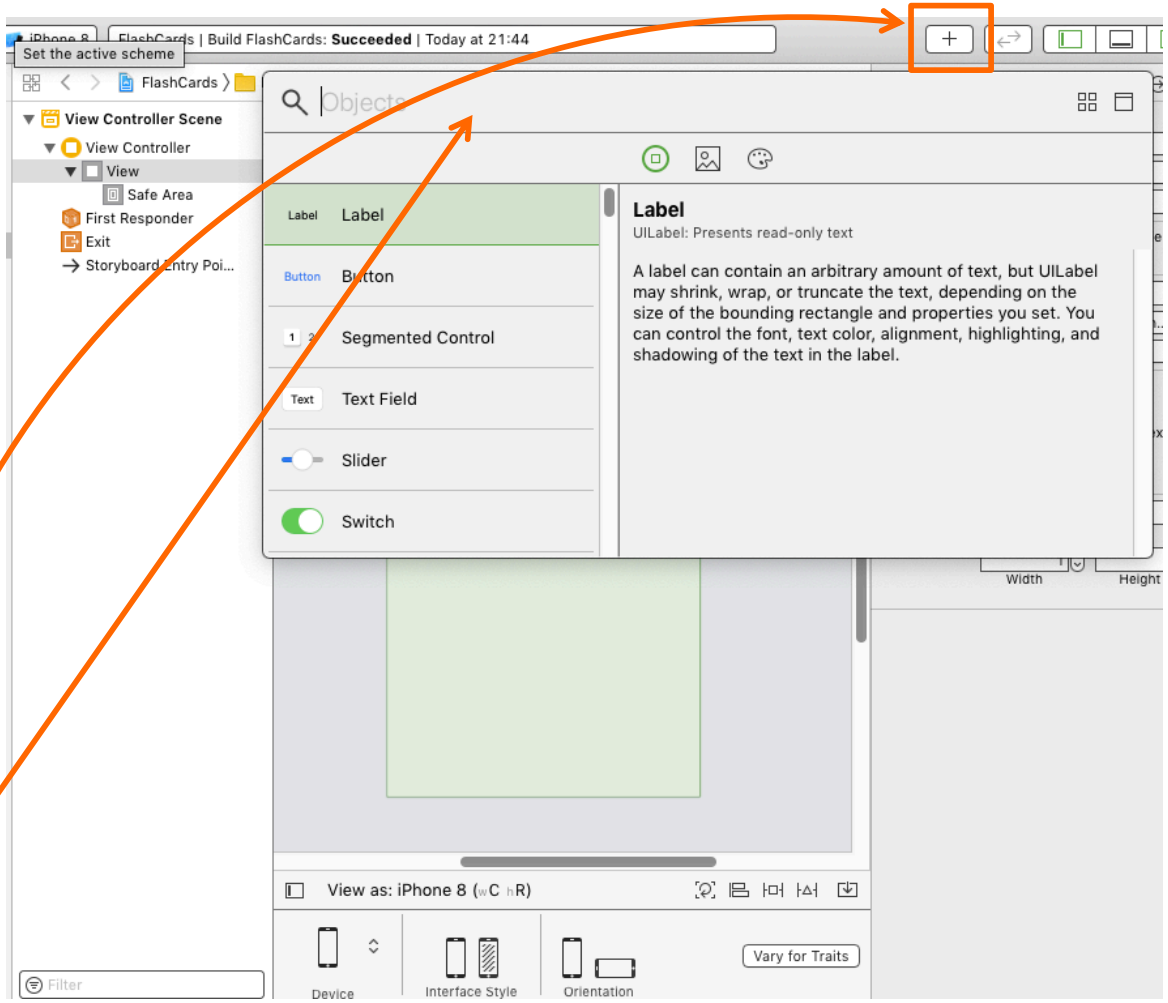
select the View object (listed in the InterfaceBuilder *dock* as part of the "View Controller"). This View object is an instance of UIView, the foundation of your app's user interface.

We'll need two additional objects in this view:

- two *labels* and
- two *buttons*.

find these in the *Library pop-up window, which you can bring up by clicking the "Library" button:* 

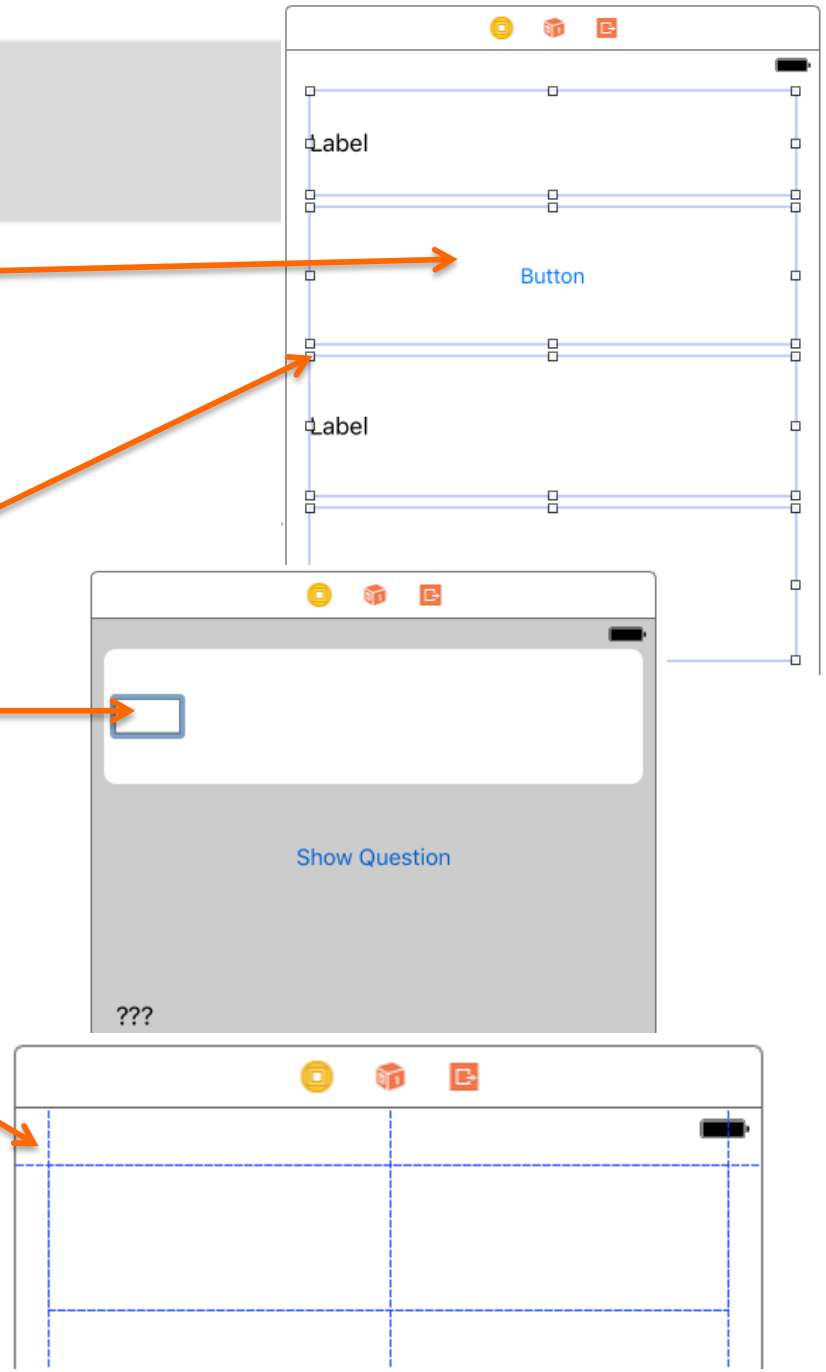
You may then use the search entry to type "button" and "label".



the View for the "FlashCards" app

- ***place*** these two *labels* and two *buttons* in the main View, then edit some of their properties directly on the canvas
- ***resize*** the *buttons* and the *labels* so that they cover most of the area
- ***edit*** the text of the two *buttons* as shown, leaving the *top label* <u>*blank*</u> – this label will be used to display a question to the user.

Always ensure that ***all*** UI elements ***align*** to the "blue dashed lines" ...this will help with *autolayout* for screen UI elements.



<u>*Note*</u>: remember to save your work in Xcode often. For example, to save the *Main.storyboard* changes, select *Main.storyboard* in the *project navigator*,then press *[command ⌘]-S* on the keyboard (or select the *File→Save* menu).

the View Controller for the "FlashCards" app

Using Xcode's *project navigator* (the leftmost pane in Xcode's window) go back to the *ViewController.swift* file and ***add*** two ***properties***, i.e. variables that will be used together with the Label view objects:

Notice:

these are Swift variables, they are of the UILabel type.

These two variables are not just "plain" variables, because.....

(this is a **first difference** from "plain" variables)

```
// we need to import the UIKit framework,
//   otherwise Swift won't know about UIViewController, etc.
import UIKit

// a subclass of the UIViewController class:
class ViewController: UIViewController {

    // two new properties for the ViewController class:
    @IBOutlet var answerLabel: UILabel!
    @IBOutlet var questionLabel: UILabel!
```

.....these variables are *prefixed* with "@IBOutlet" – they're *IBOutlet*s

Here's **why we use IBOutlet**s here:

Interface Builder scans your source code looking for any properties in your view controller prefixed with the "@IBOutlet" keyword. Xcode then exposes to Interface Builder any "@IBOutlet" properties it discovers, so you'll be able to ***connect*** those ***variables*** to ***views*** in Interface Builder.

the View Controller for the "FlashCards" app

Second difference from "plain" variables:
these variables are defined with an exclamation mark (!) appended to their type.

```
// we need to import the UIKit framework,
//   otherwise Swift won't know about UIViewController, etc.
import UIKit

// a subclass of the UIViewController class:
class ViewController: UIViewController {

    // two new properties for the ViewController class:
    @IBOutlet var answerLabel: UILabel!
    @IBOutlet var questionLabel: UILabel!
```

Here's why: the (!) sign *appended to the variable type* indicates that these variables have ***optional values***, but they are "*implicitly unwrapped*" ...***what*** does this ***mean?!?***

Practically speaking, this means that you can write Swift code *assuming* that these variables will have a set value by the time you use them .... but your app will *crash* if you try using such a variable *before* it has value!

In Swift these types are called ***implicitly unwrapped optionals***, i.e. those having an (!) sign at the end of their type definition. Here in the "***FlashCards***" *ViewController.swift* code, ***we use optionals*** to define variables that will be ***certainly set up before being used***, e.g. for user interface elements we create in the Storyboard, even though they don't exist at first, when the app first starts running, before the main View shows up on the screen.

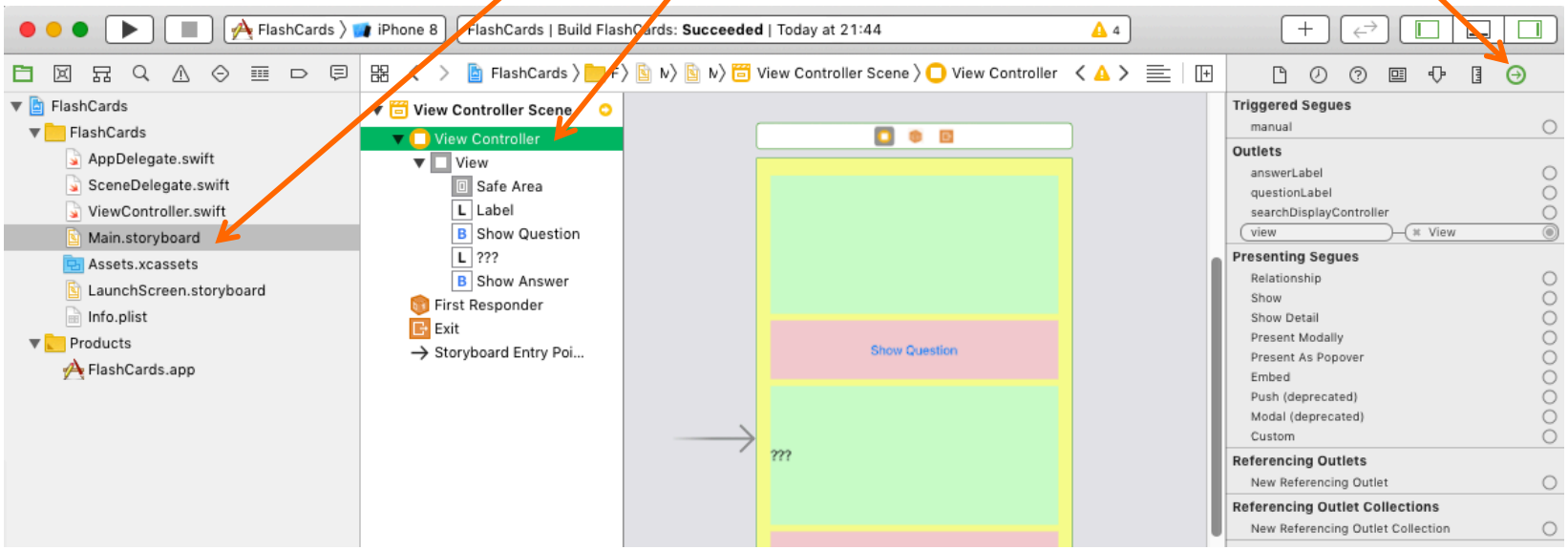
(for a bit more details, "**optionals**" in Swift are explained here: https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html - ID330)

the View Controller for the "FlashCards" app

Now to ***connect*** *properties* (i.e. the two variables we just defined in ViewController.swift) *to the user interface elements*

1. in Xcode's *project navigator* select **Main.storyboard**, then
2. in Interface Builder's *dock* select **View Controller**, then
3. in the Inspector, select Connections (***the 7th tab***).

You'll see that *the IBOutlet properties you created in ViewController.swift* are now listed in the Outlets section:

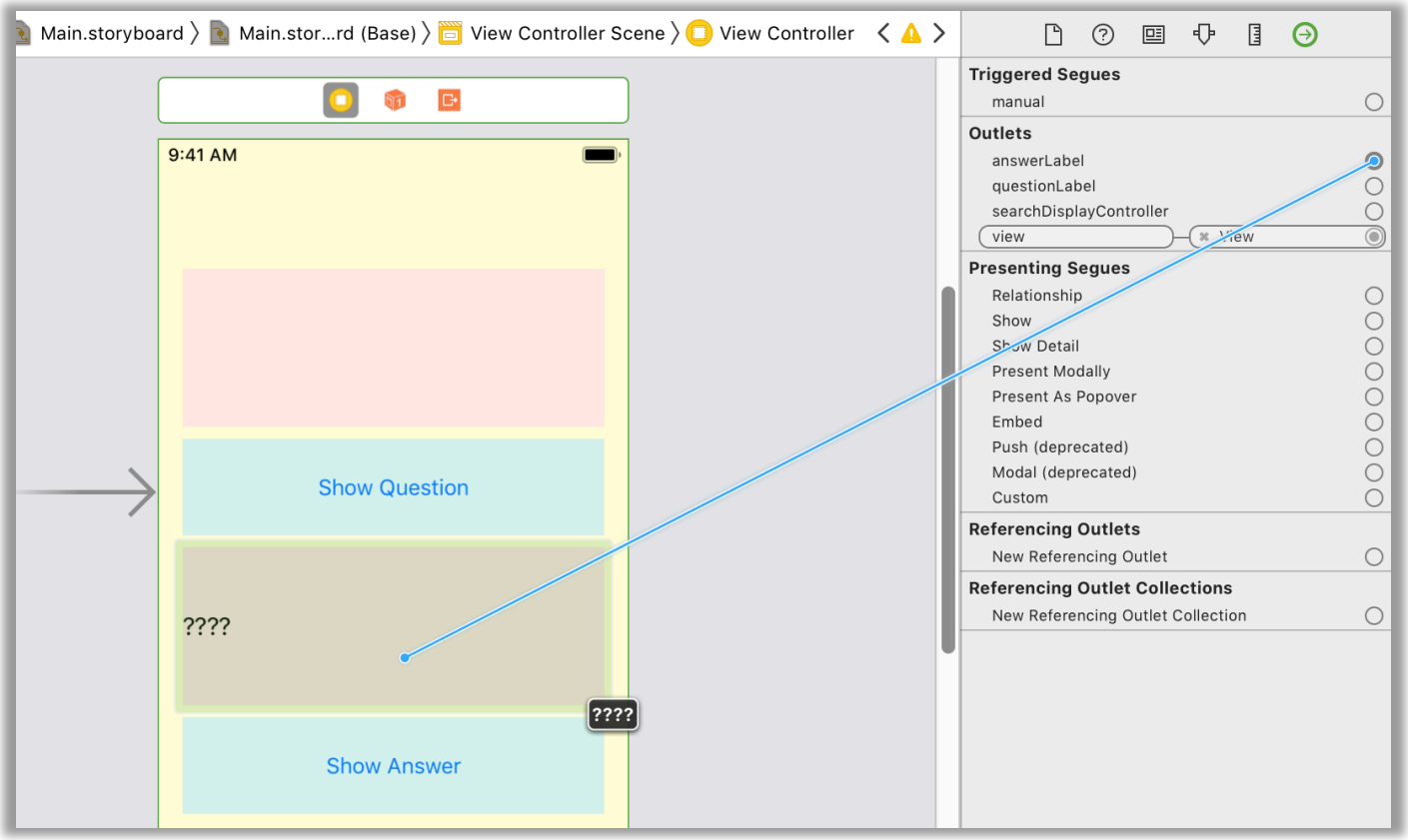


the View Controller for the "FlashCards" app

In the "outlets" listed in the Connections tab, there is a *circle* on the right of every outlet. For example, there is a circle on the right of *answerLabel*.

Using the mouse, ***drag*** (i.e ***click*** the mouse button, then ***drag*** the mouse) from the circle on the right of *answerLabel* in the inspector, to the"???" *label* above the Show Answer button in the Canvas, then release the mouse button to ***connect*** the"*answerLabel*" Swift property to the "???" *label*.

Do this as well: complete a similar click-drag connection for the *questionLabel*: ***connect*** it to the *label* above the Show Question button in the Canvas.



the View Controller for the "FlashCards" app

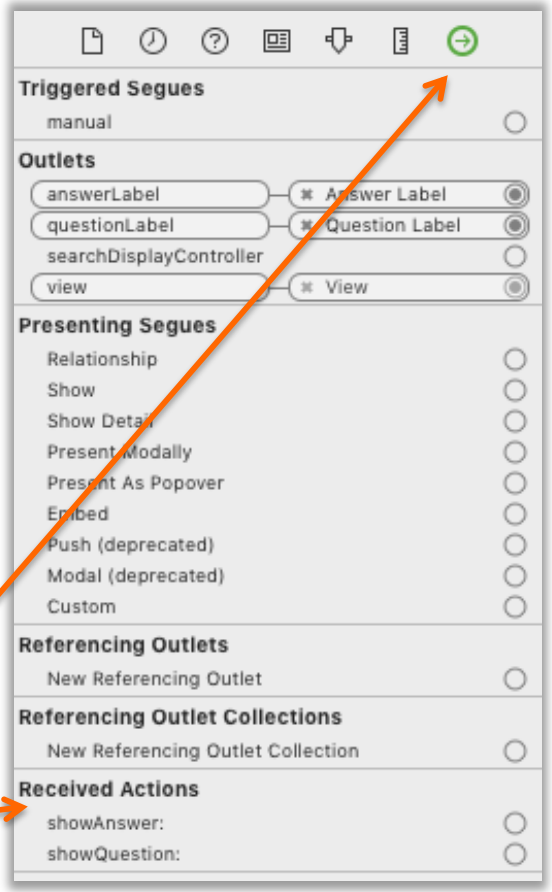
Then, to connect the two buttons in the view to your *controller*,

type Swift code to define the two @IBAction methods inside your *ViewController.swift* file:

```
class ViewController: UIViewController {

    // add two new properties to the ViewController class:
    @IBOutlet var answerLabel: UILabel!
    @IBOutlet var questionLabel: UILabel!

    // two methods to handle button events:
    // @IBAction here tells Xcode to list these methods as
    //    "available actions", and they can therefore be connected
    //    using InterfaceBuilder, to buttons in Main.storyboard:
    @IBAction func showQuestion(_ sender: Any){
    }
    @IBAction func showAnswer(_ sender: Any){
    }
```

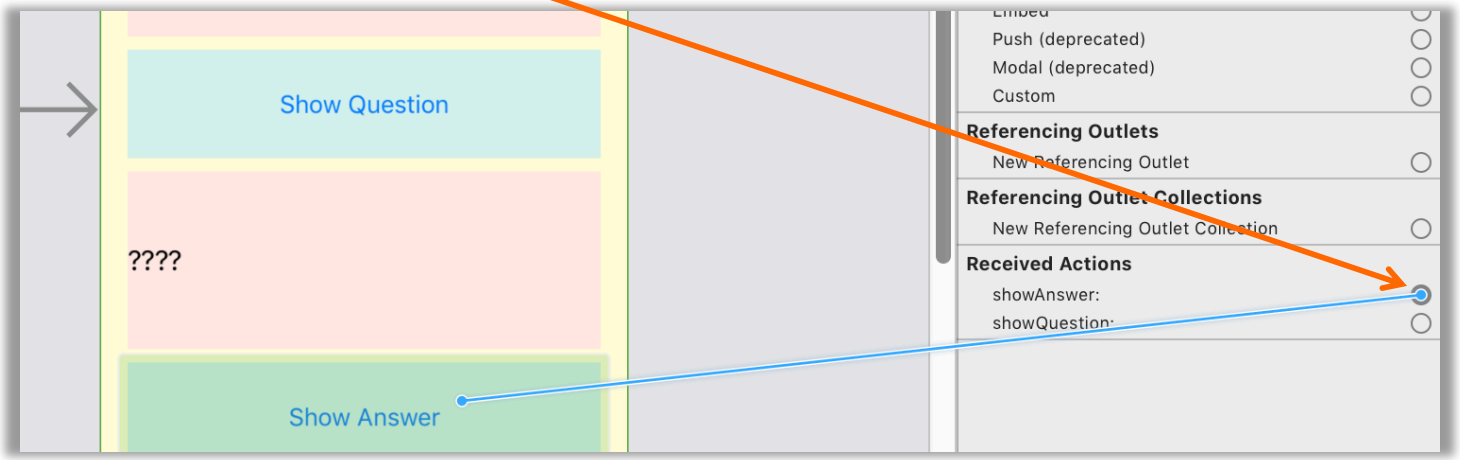


the "@IBAction" keyword is used by Xcode to then provide these methods in the ***Received Actions*** section in the *Connections* tab of the *Inspector* *(Xcode's right-side pane)*

the View Controller for the "FlashCards" app

The *ViewController.swift* file now contains two new methods to interface to the buttons in the View.

Connect (*) these methods from the *Received Actions* section of the *inspector* to the View elements in the Canvas, i.e. the two buttons "Show Answer" and "Show Question":

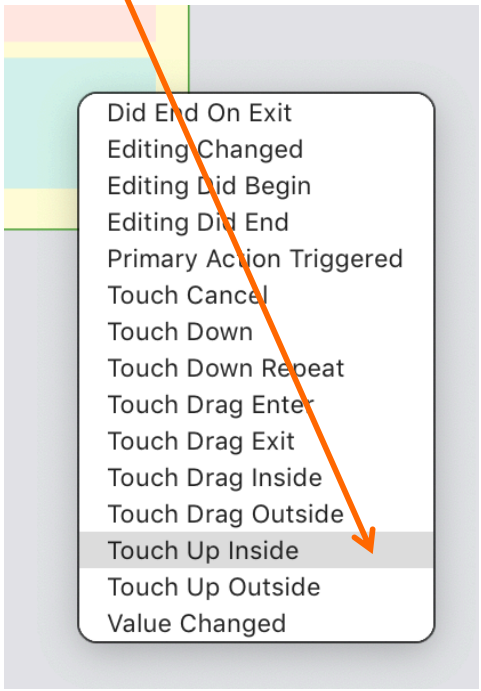


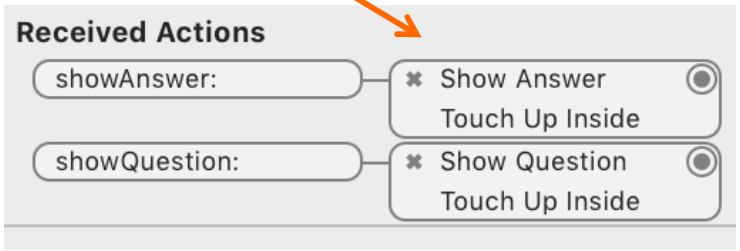
(*) just as you earlier connected the two Outlets to their View elements, but this time... (continue on next slide)

the View Controller for the "FlashCards" app

(continues from previous slide) ... you will also need to ***select*** "Touch Up Inside" in the pop-up menu that shows up when you connect the Received Actions to their respective Views.

Select this for both methods/buttons, so that the connection will show up as "Show Answer" and "Touch Up Inside" in the inspector





testing your "FlashCards" app so far

to make sure that everything is updated in your App executable, while in Xcode

press *[command ⌘]-[Shift ⇧]-K* on the keyboard (or select the *Product→Clean Build Folder* menu), to remove compiled/executable code from your project, then

press *[command ⌘]-R* on the keyboard (or select the *Product→Run* menu), to run your freshly compiled app in the iOS simulator.

While the app has no functionality yet, the buttons will show that they "work" by changing their appearance when you *tap* (or while in the iOS Simulator, when you *click*) them.

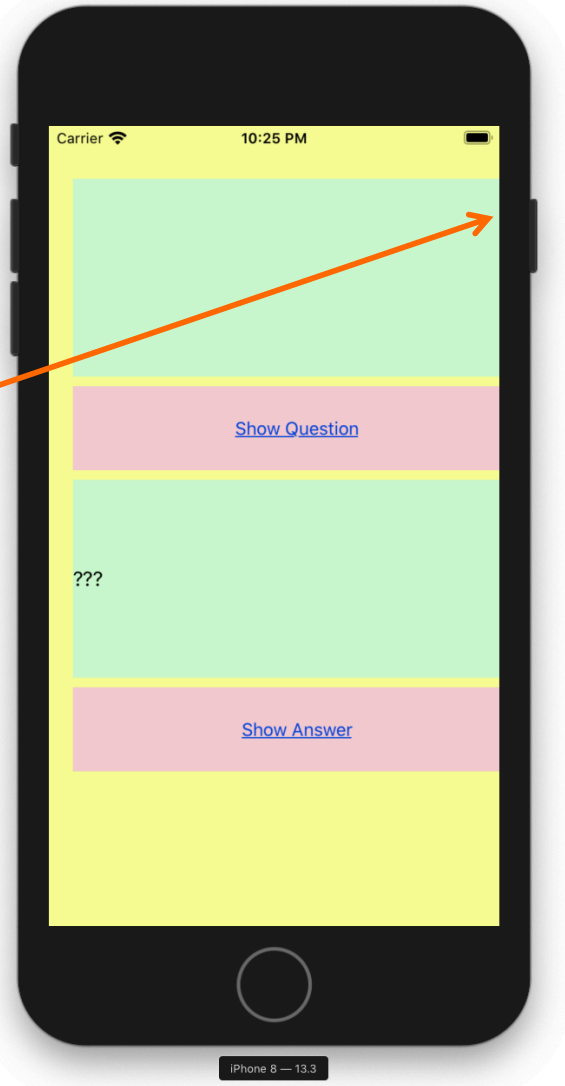
testing your "FlashCards" app so far

it all works!

Or does it...?

The "autolayout" feature may not quite work as we might have expected from having aligned all the "dotted blue lines" in Interface Builder elements.

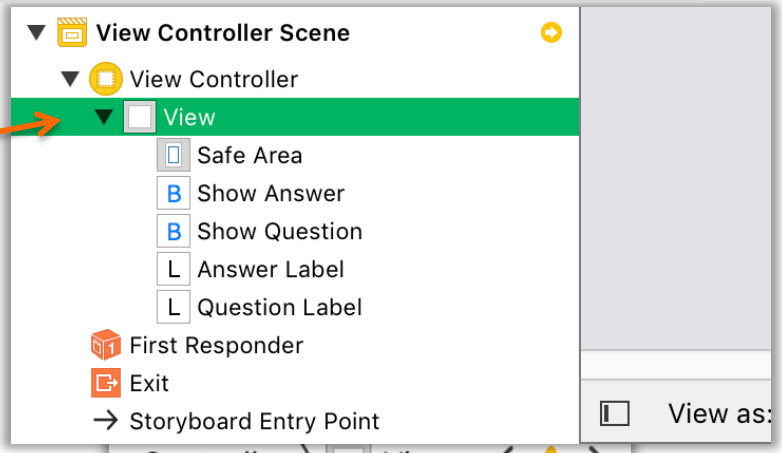
Another step is necessary for the "autolayout" to do anything: set the ***constraints*** between user interface elements: we'll do that on the next slide.

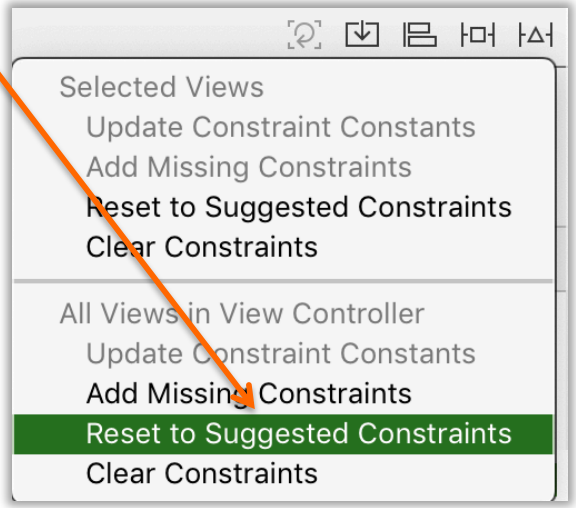


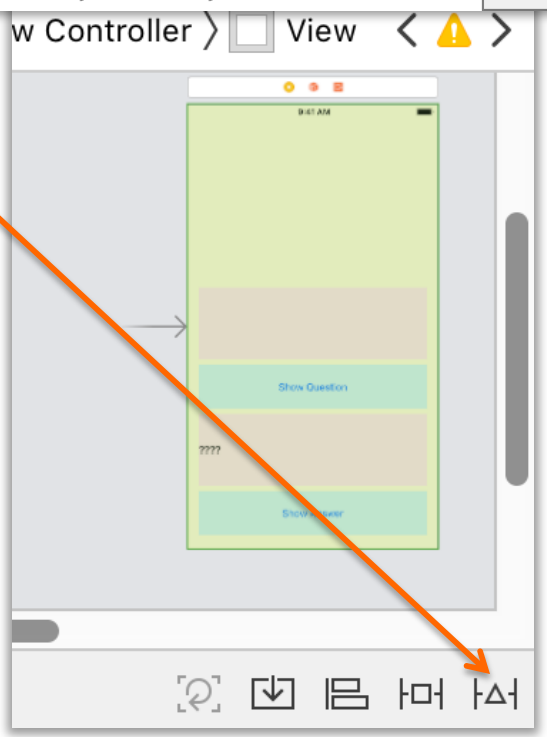
testing your "FlashCards" app so far

We'll ask Interface Builder to set "autolayout" for our app, to see if "Suggested Constraints" may be helpful:

1. select **View** in the Interface Builder's Dock
2. click the "*Resolve Auto Layout Issues*" button (the one that looks like a triangular-shaped *Tie-fighter* ?)
3. select "***Reset to Suggested Constraints***" in its pop-up menu for "*All Views in View Controller*".



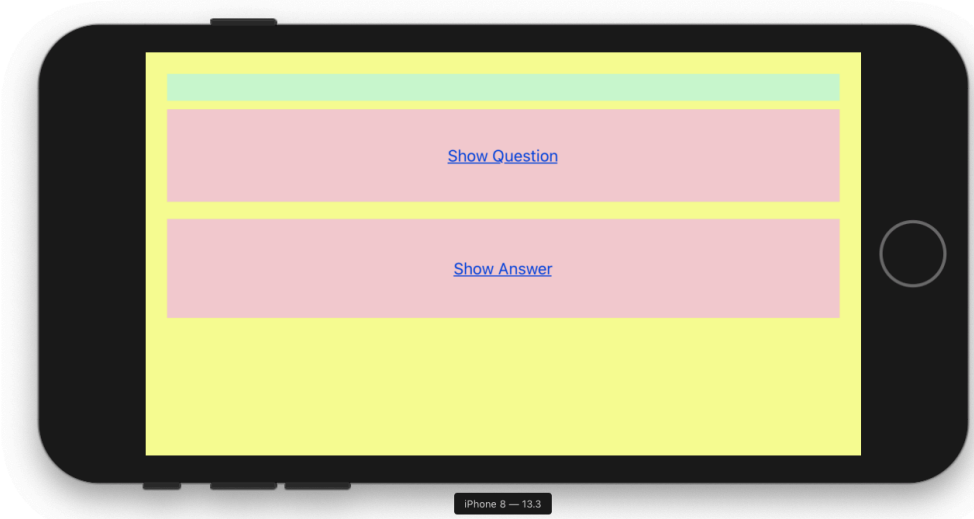




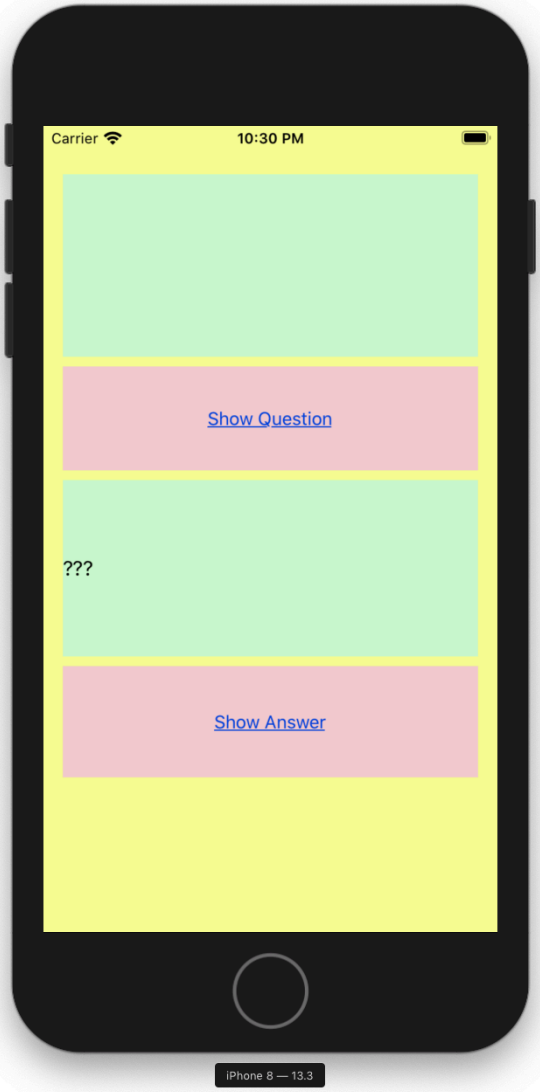
testing your "FlashCards" app so far

OK, perhaps this app –as it is so far- won't win a Design Award this year... but at least buttons and labels show up on the screen in vertical mode.

(here shown after extra non-standard colors were added for clarity)



However, as shown above here↑, when the app runs in horizontal mode, things are still not quite solved...



adding some "behavior"

to add some visible "behavior" to our app,
(even before writing any Model code)
we'll add some code to the two "@IBAction" functions that were left empty before:

```
// two new properties for the ViewController class:
@IBOutlet var answerLabel: UILabel!
@IBOutlet var questionLabel: UILabel!

// two methods to handle button events:
// @IBAction here tells Xcode to list these methods as
//     "available actions", and they can afterwards be connected
//     using InterfaceBuilder, to buttons in Main.storyboard:
@IBAction func showQuestion(_ sender: Any) {
    self.questionLabel.text = "How old are you?"
    self.answerLabel.text = "(...try guessing...)"
}
@IBAction func showAnswer(_ sender: Any) {
    self.answerLabel.text = "I'm almost 6 years old! (that was the
        Swift programming language's answer)"
}
```

compile and run the app again, and test it by pressing on either one of the two buttons in the User Interface.

running the app with some "behavior"

here are the results of our efforts:

There's One More Thing... *to do for HW 01!*

- there's ***one more thing*** to do
  in order to complete Homework 01:

  → ***write*** the code to ***correct*** an incorrect behavior in our app:

  → as things are right now,
  the @IBAction methods in the *ViewController*,
  cause the FlashCards app to *show an* ***answer***
  every time we tap on the "*Show Answer*" button...
  ...even if *no* ***question*** has been shown yet!

- To complete Homework 01,
  you need to ***modify*** your FlashCards app...

  (side note: think about where this behavior should be achieved?)

- ... so that it will ***not*** *show any* answer
  until a question has been shown.

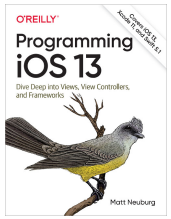
for Homework 01, turn in your "*FlashCards*" Xcode project to IU GitHub

1. complete, compile, and run the "***FlashCards***" app in the iOS Simulator
2. when you're done:
   - **a.** **quit** the iPhone Simulator
   - b. select the Xcode menu *Product* → *Clean Build Folder*
   - **c.** **quit** Xcode, then find your "***FlashCards***" folder in the macOS Finder
   - **d.** **rename** your "*FlashCards*" folder to "FlashCards-*yourusername*"
   - **e.** **write** a README plain-text file named
     'hw01-README-C323-*yourusername*.txt'
     in which you explain:
     i. the parts of Homework 01 you have completed
     ii. any interesting parts you may have added
     iii. (optional) any question you may have about the tasks,
     and any issue you may have encountered with the software tools you used
3. turn in your own submission to your personal IU GitHub repository for C323, under **/hw/hw01/** (or similar, e.g. /homework/homework01/ as long as you are consistent)
   (For setting up IU GitHub, please review Lab 02 notes!)
4. the submission for Homework 01 needs to contain your complete "FlashCards-*yourusername*" Xcode project folder (it should ***not*** be compressed/zipped: every file should be on IU GitHub individually)

(*clarification*: for Homework 01, there is nothing to submit directly on IU Canvas:
the entire Homework 01 submission is to your own personal IU GitHub repository for C323)

Note: in the above instructions, *yourusername* is your IU username, not the word "yourusername" !

References

1. M-V-C in Xcode:
   pages 285-287 in the 

   textbook

2. UML:

   "class diagram in the Unified Modeling Language (UML) "
   https://en.wikipedia.org/wiki/Class_diagram

3. developer.apple.com – **Xcode** – Interface Builder Connections Help:
   - Add an ***action connection*** to receive messages from a UI object:
     https://help.apple.com/xcode/mac/10.2/#/dev9662c7670
   - Add an ***outlet connection*** to send a message to a UI object:
     https://help.apple.com/xcode/mac/10.2/#/devc06f7ee11

4. Optionals in Swift:
   https://docs.swift.org/swift-book/LanguageGuide/TheBasics.html#ID330