CS275
Fall 2020

iOS: The Camera

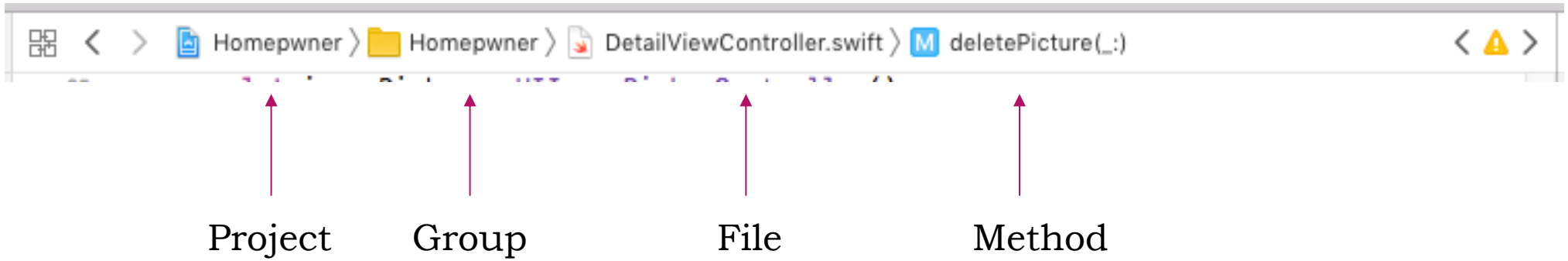
BNRG CHAPTER 15

Topics

- Using the camera in an app
- `UIImageView`
- `NSCache` - sophisticated way for iOS to cache data for apps

Jump Bar in Xcode

First, a quick discussion of the "source editor jump bar"



Click on any of these, and you'll see a popover of other parts of the project at that level of the hierarchy

Useful Xcode Feature: // MARK:

Put this into the source file to organize the presentation of methods in the popover for a file

```
// MARK: -
```

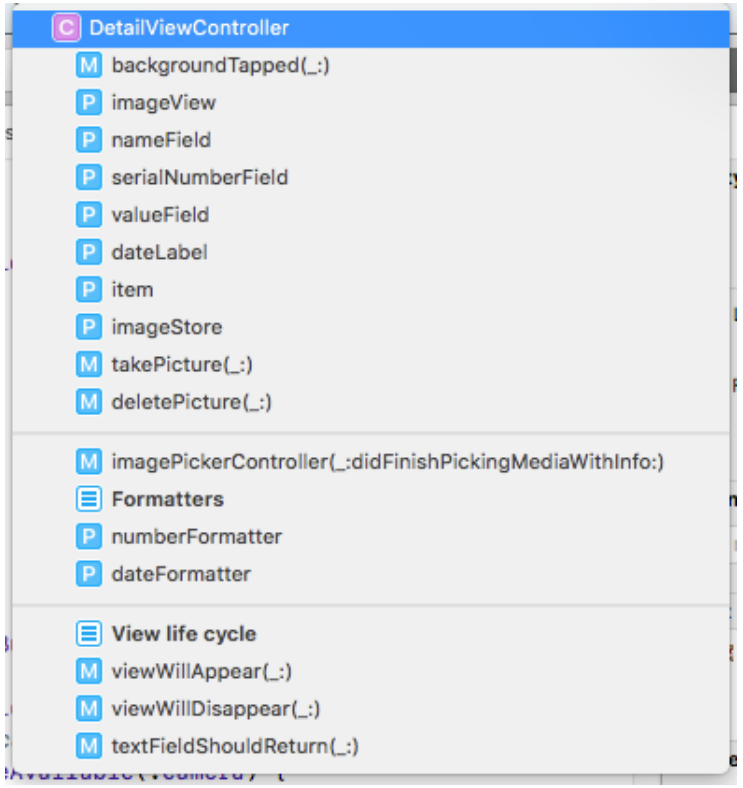
Can also use a label

```
// MARK: Formatters
```

And combine them

```
// MARK: - View life cycle
```

This does not change the code—it just tells Xcode how to visually present the methods in your file



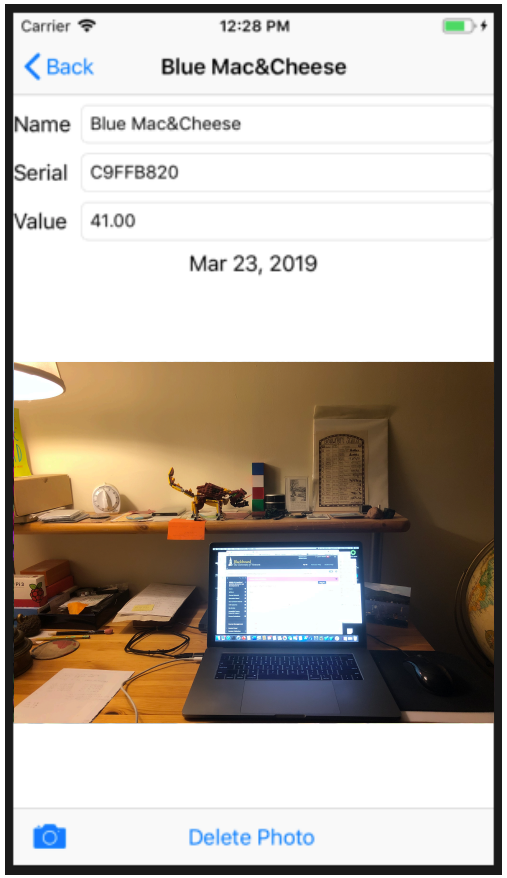
Images

In the overview-detail view, we can add an image to the detail view

- the image will be a photo, provided by the camera

We will display photos in a `UIImageView`

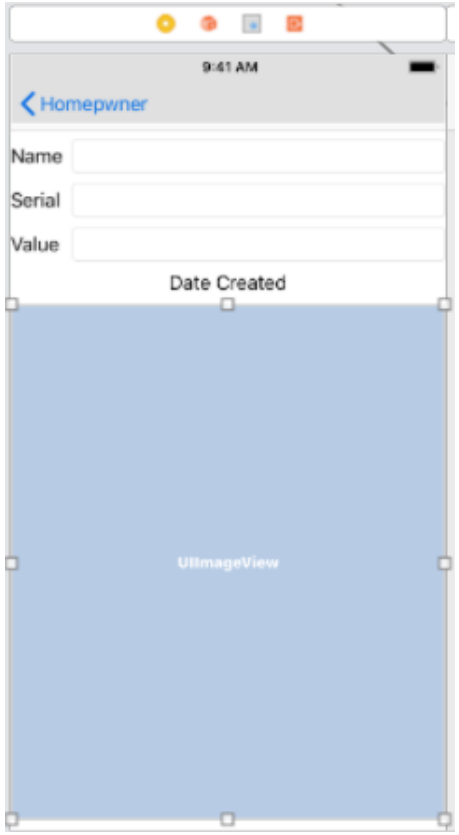
We'll save the images using ImageStore



Adding a `UIImageView`

An easy way to display an image in an app is in a `UIImageView`

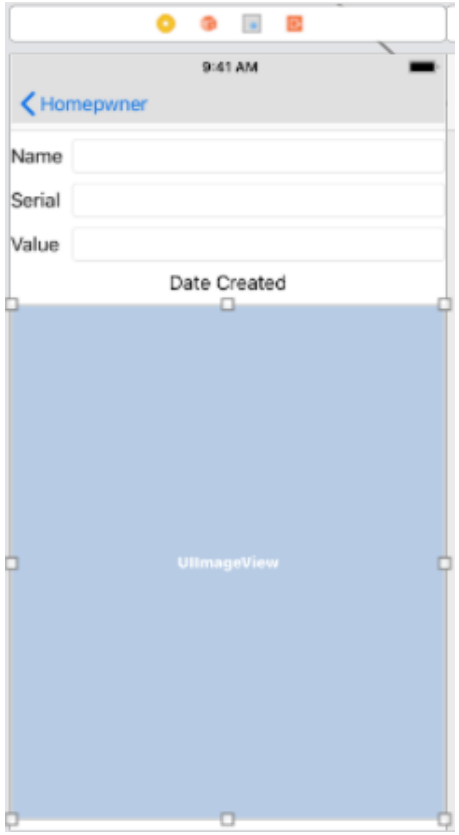
In the detail view, we can add a `UIImageView` to the vertical stack view in `DetailViewController`



Adding a `UIImageView`

To have the image fill as much space as possible, change the content hugging and content compression-resistance priorities

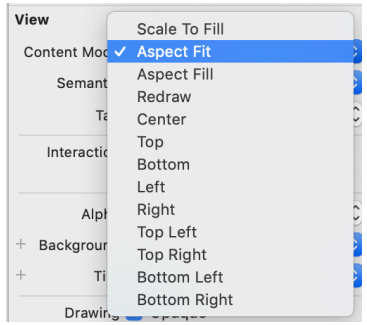
- content-hugging priority: like a rubber band around a view; higher value means stronger rubber band (which prevents the view from expanding beyond its intrinsic size)
- content-compression-resistance priority: like springs on the inside of the view; higher value means stronger springs (and so the view resists compression)
- make the vertical CH priority smaller than the default (which is 250)
- make the vertical CCR priority smaller than the default (which is 750)



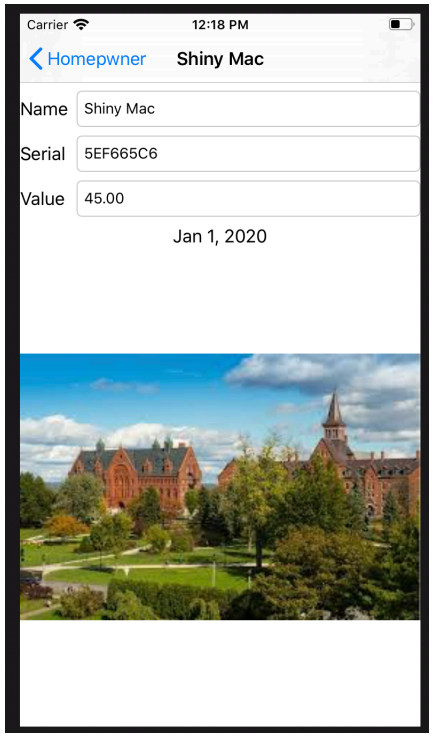
`UIImageView` Display Mode

A `UIImageView` displays its contents according to its `contentMode` property

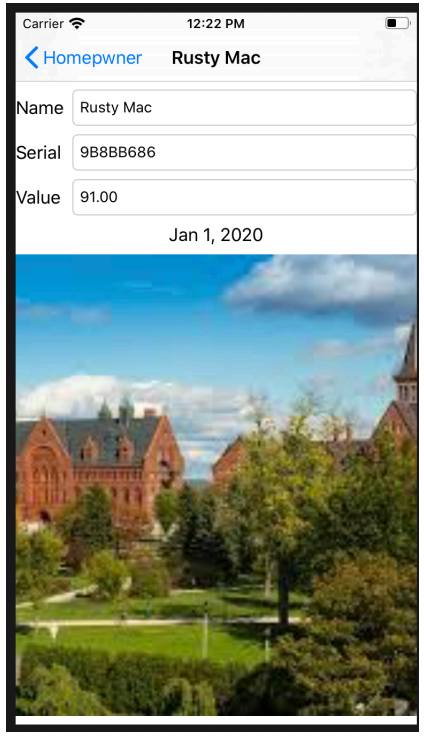
- this property determines where to position and how to resize the content within the image view's frame
- `UIViewContentMode.scaleToFill`: adjusts the image to match exactly the bounds of the image view
- the default is `.scaleAspectFill`, which keeps its aspect ratio unchanged in order to center the image in the image view
- other option: `.scaleToFill`, which changes the aspect ratio if necessary in order to have the image match the bounds of the image view
- can set this programmatically or on the storyboard



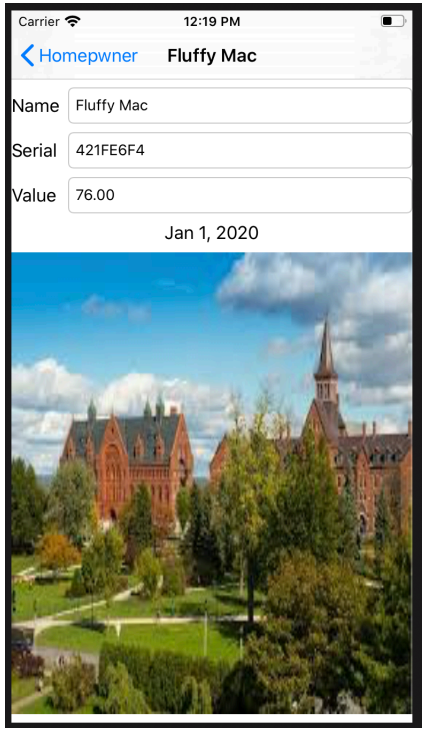
UIImageView Display Modes



.scaleAspectFit



.scaleAspectFill



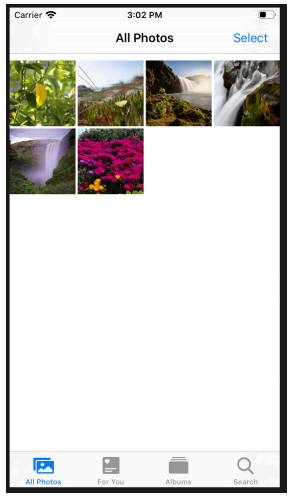
.scaleToFill

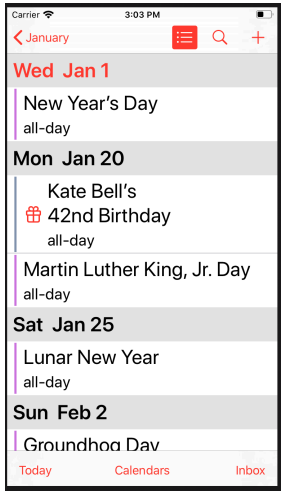
UIToolbar

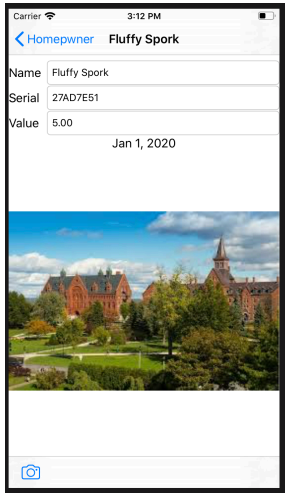
UIToolbar: a control that displays one or more buttons along the bottom edge of an interface

- UIToolbar has an array of bar buttons (UIBarButton)

For this app, the toolbar is a good place to add a camera button, to let the user take a photo







examples of UIToolBar

Getting an Image from the Camera

Main interface to the camera and to the photo album: `UIImagePickerController`

- specify a source type, and the `UIImagePickerController` does the rest

UIImagePickerController

`UIImagePickerController` specifies the source of an image

- from the camera (lets the user take a new photo)
- or from an album (prompts the user to pick an album and then a photo from that album)
- or from recently taken photos

To use `UIIImagePickerController`, specify a `sourceType`

- `UIImagePicker.SourceType.camera`: use the camera
- `UIImagePicker.SourceType.photoLibrary`: from an album
- `UIImagePicker.SourceType.savedPhotosAlbum`: from recently taken photos

UIImagePickerController

To let the user take a new photo

- first query whether the device has a camera—this is just general good practice as part of "defensive programming"
- the method to do this is: `isSourceTypeAvailable(_:)` on the `UIImagePickerController` class

UIImagePickerController

Checking to see whether the device has a camera

```
@IBAction func takePicture(_ sender: Any) {
    let imagePicker = UIImagePickerController()
    // if the device has a camera, then take a photo;
    // otherwise, pick from photo library
    if UIImagePickerController.isSourceTypeAvailable(.camera) {
        imagePicker.sourceType = UIImagePickerController.SourceType.camera
    } else {
        imagePicker.sourceType = .photoLibrary
    }
}
```

UIImagePickerControllerDelegate

`UIImagePickerController` also needs a delegate

- when the user selects a photo from the `UIImagePickerController's` interface, the delegate is sent the message `imagePickerController(_:didFinishPickingMediaWithInfo:)`
- if the user cancels the action, then the delegate receives the message `imagePickerControllerDidCancel(_:)`

Easy way to do this:

- have `DetailViewController` conform to `UIImagePickerControllerDelegate`
- and set the `imagePicker`'s delegate property to `self`
- must also have `DetailViewController` conform to `UINavigationControllerDelegate`, since `UIImagePickerControllerDelegate` references an object that conforms to `UINavigationControllerDelegate`

Present the Image Picker

After creating an instance of `UIImagePickerController`

- put the VC up on the screen modally (modally: forces a response from the user):

```
present(imagePicker, animated: true, completion: nil)
```

Simulator: No Camera

Unfortunately, the device simulator does not provide support for a camera

- some people have posted a sort-of workaround to this on Stack Overflow

But to take photos in this project, you'll have to connect an Apple device

- otherwise, you can select one of the stock photos that are on the simulator

Permissions

First try: clicking the camera button to take a photo:

```
2020-01-01 16:29:51.348492-0500 HomePwnerCh15[1573:146064] [access] This app
has crashed because it attempted to access privacy-sensitive data without a
usage description.  The app's Info.plist must contain an
NSCameraUsageDescription key with a string value explaining to the user how
the app uses this data.
```

Permissions

Several capabilities of an iOS device require explicit user permission before an app can use them:

- camera and phone
- location
- microphone
- HealthKit data
- calendar
- reminders

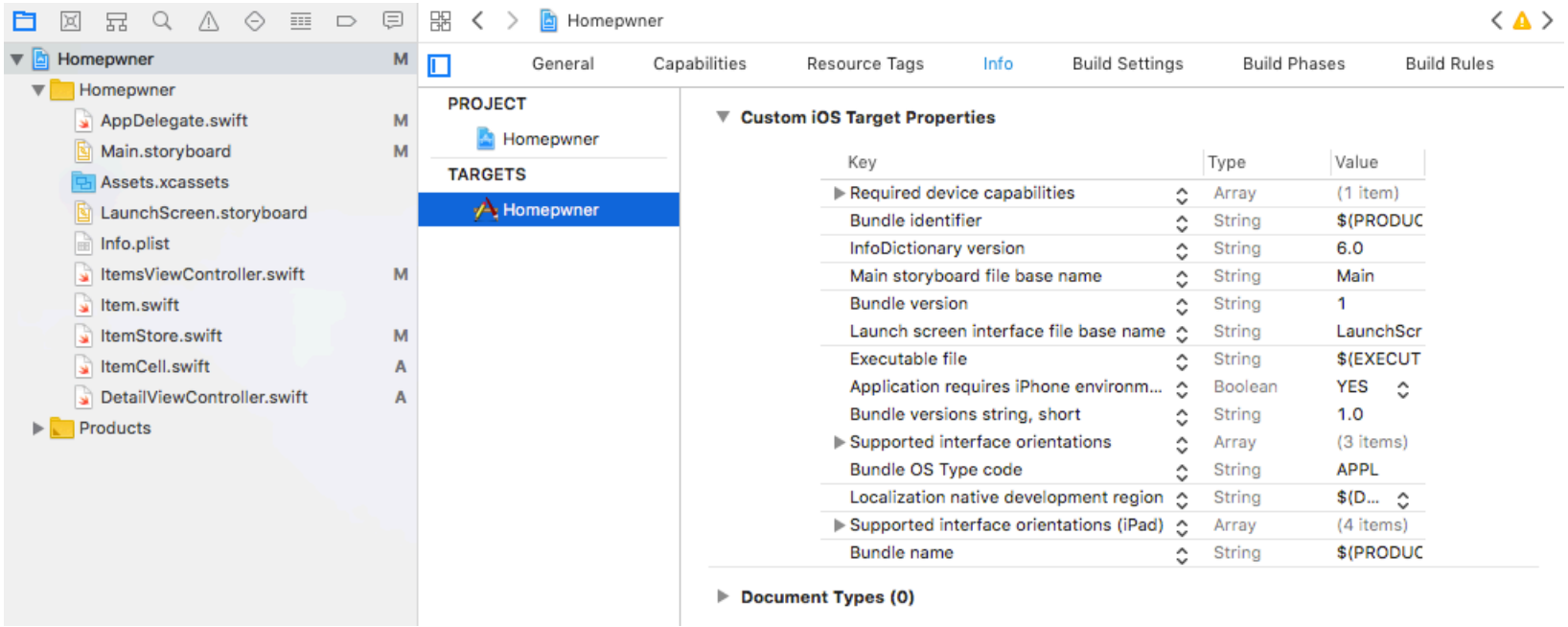
Permissions

In order to use one of the capabilities, an app must first ask for permission

- and in order to ask for permission, the app must supply a usage description that specifies the reason why the app wants to use a particular capability
- that description will then be presented to the users whenever the application accesses that capability

Project Info

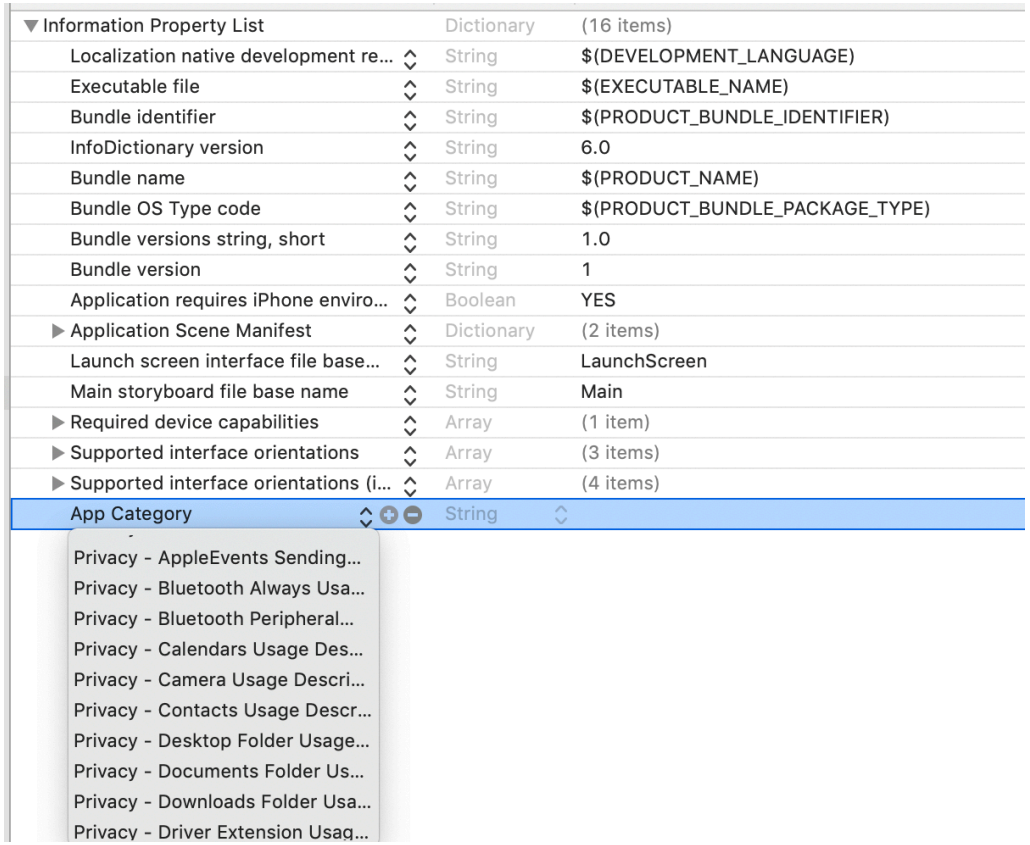
The usage description will go into the project info for the app



Camera Usage Description

The procedure for adding a usage description for the camera is a little different from what the book has

- the key you'll look for is "Privacy – Camera Usage Description"
- behind the scenes, this information goes into a file called `Info.plist`, in your project
- for the photo library, look for "Privacy – Photo Library Usage Description" in the list



Camera Usage Description

Note: the iOS simulator doesn't actually ask for permission to access the photo library

- this is either a bug or a quirk of the simulator
- or maybe it's because no one cares whether the simulator accesses the stock photos that it comes with :-/

But if you run this on a real device, you should see the permissions dialog

Saving the Image and Displaying It

When the `UIImagePickerController` finishes, it calls a method on its delegate (in this case, the DetailViewController)

The interface of the method on p. 271 has changed

```
func imagePickerController(_ picker: UIImagePickerController,
                           didFinishPickingMediaWithInfo info: [UIImagePickerController.InfoKey : Any]) {
    // get picked image from dictionary
//  let image = info[UIImagePickerController.originalImage] as! UIImagePickerController
    if let image = info[.originalImage] as? UIImage {
        // put that image on the screen in the image view
        imageView.image = image
    }
    // take image picker off the screen -- must call this dismiss method
    dismiss(animated: true, completion: nil)
}
```

https://stackoverflow.com/questions/51342028/cannot-subscript-a-value-of-type-string-any-with-an-index-of-type-uiimage

Saving the Image: Refinement

Instead of saving an image from an item directly in a property (i.e, in memory)

- it's better to save in a special cache in storage (NVM = nonvolatile memory: the storage for the device)
- for example, if there were dozens of items in the app, then the memory required to keep dozens of photos in memory (one for each item) could be a problem
- instead, the cache keeps the image in NVM and brings it into memory only when the app asks to display it

To do this, we create a new class (called `ImageStore`) that uses `NSCache`

- this class will encapsulate the management of the images
- and eventually manage the writing and reading of image files

NSCache

`NSCache`: an Objective-C implementation of a dictionary

- but even better: it will automatically remove objects if the device gets low on memory

`NSString`: an Objective-C implementation of a string

- must use this for the key since this is what `NSCache` uses for its key

ImageStore

Basic idea: give each Item a unique identifier (a `UUID`: a unique value that is easily created)

- use this unique identifier to form a key
- store the image in an `NSStore`, using this unique identifier as a key

When the detail view for the item will appear, retrieve the image from the cache

- if the item is deleted, delete the image from the cache

Encapsulate the cache activities in a new class: `ImageStore`

- this solution is OK for now, but it's a short-term solution
- the better solution will be to save the each image in a file, which the app will manage

ImageStore

```
class ImageStore {
    let cache = NSCache<NSString, UIImage>()

    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)
    }
}
```

Access to the `ImageStore`

The `DetailViewController` needs access to an instance of `ImageStore`

- solution: create an `ImageStore` up front, during app initialization (same as with the `itemStore`)