Lecture 5

Road Map

- More Swift: *In-Out parameters, OptionSet*
- View Controller Lifecycle
- Segues
- Demo: *delegation*

In-Out Parameters

We can’t modify the value of a function parameter within the body of the function, they are constants by default.

```
1
2 func foo(aParameter: Int) {
3     aParameter += 1
4 }
```

Left side of mutating operator isn't mutable: 'aParameter' is a 'let' constant

BUT! In-out parameter can be changed within the body of the function, and that changes persist after the function call is ended. Only variables can be passed as in-out parameters, as they can be modified.

To declare an in-out parameter, place the ***inout*** keyword before the parameter’s type:

```
func foo(aParameter: inout Int) {
    aParameter += 1
}
```

To pass a variable into a function, place a “**&**” sign before the variable’s name

```
var someInt = 21
foo(aParameter:  &someInt)
```

OptionSet

This is a **protocol** to represent bit mask types, where each *bit* represents members of the set.
When you create an option set, you include a rawValue property.
This property must be of a type that conforms to the BitwiseOperations protocol (e.g. Int).

```
struct DeliveryOptions: OptionSet {
   let rawValue: Int
}
```

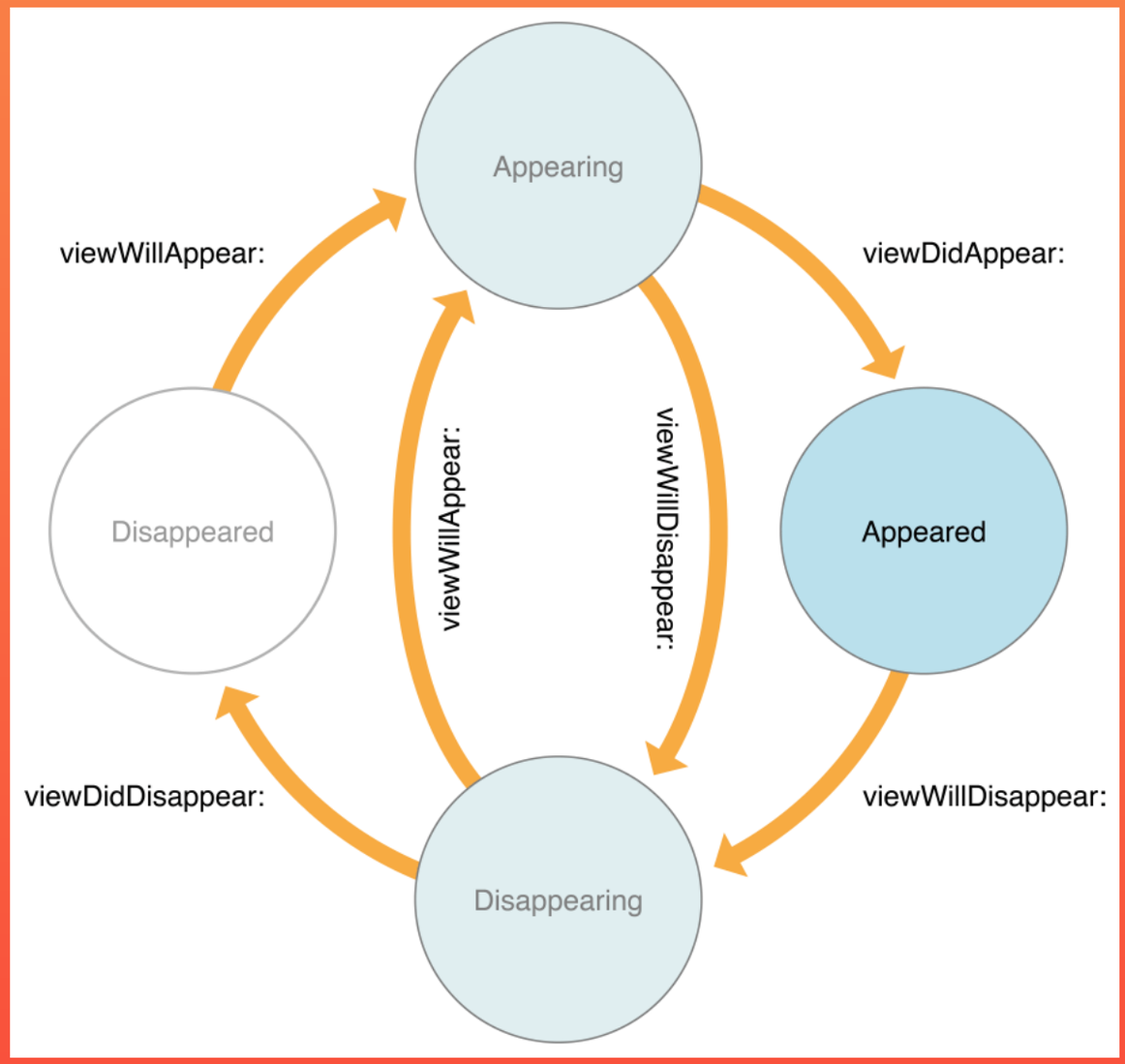
To create a new option, declare a static property of your custom type using unique powers of two (1, 2, 4, 8 …) for each property's raw value (each property then can be represented by a single bit of the type's raw value)

```
struct DeliveryOptions: OptionSet {
   let rawValue: Int

   static let standard = DeliveryOptions(rawValue: 1 << 0)
   static let premium = DeliveryOptions(rawValue: 1 << 1)
   static let express = DeliveryOptions(rawValue: 1 << 2)
}
```

View Controller Lifecycle

— a set of methods that each object of UIViewController class or its subclasses possesses. These methods are called automatically by a system during the controller's lifetime. When you create a subclass of a view controller, these methods are inherited from the UIViewController, so you can add your custom behaviour for each method.



Source: developer.apple.com

View Controller Lifecycle

viewDidLoad – is called when the view controller's content view (the top of its view hierarchy) is created and loaded from a storyboard.
The outlets are guaranteed to be set up by the time this method is called. This is a very good place to do any additional setup (e.g. update your UI from the Model).

```
func viewDidLoad() {
   super.viewDidLoad()
   // your setup
}
```

But note that by that time the view's geometry is not set yet, so do not initialise any things that depend on view's geometry.

View Controller Lifecycle

viewWillAppear – This method is called just before the view controller's content view appears on screen (more precisely – before it is added to the application's view hierarchy).

Note that unlike "viewDidLoad" method, which is called only once when the view has actually loaded, this method can be called several times because view might appear and disappear a lot.

Note that despite the name of the method, the system doesn't guarantee that the content view will become visible because it may be hidden or obscured by other views. Technically this method indicates that the view controller's content view to be added to the application's view hierarchy.

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

    //your code here
}
```

View Controller Lifecycle

viewDidAppear – similarly this method is called just after the view controller's content view has appeared on screen (more precisely – after it has beed added to the application's view hierarchy).
You should use this method to trigger any actions which require the view to be presented on screen (e.g. data fetching [to optimise performance] or showing an animation)

Also you will be notified with the similar set of methods (**viewWillDisappear** and **viewDidDisappear**) when the view is to disappear off screen.

```
override func viewDidAppear(_ animated: Bool) {
    super.viewDidAppear(animated)
    //…
}
override func viewWillDisappear(_ animated: Bool) {
    super.viewWillDisappear(animated)
    //…
}
override func viewDidDisappear(_ animated: Bool) {
    super.viewDidDisappear(animated)
    //…
}
```

View’s geometry changes

When a view’s bounds change, the view adjusts the positions of its subviews.
The view controller is notified about these events by **viewWillLayoutSubviews()** and **viewDidLayoutSubviews()** methods, which are called just before and just after the view lays out its subviews respectively.

```
override func viewWillLayoutSubviews() {
   //default implementation does nothing
}

override func viewDidLayoutSubviews() {
   //default implementation does nothing
}
```

Note that most of the time you will use **Autolayout** (will speak about it in a couple of weeks) where most of the stuff is handled for you automatically.

Memory Management

Memory is a critical resource in iOS. When the system determines that the the amount of the available memory is low, the method **didReceiveMemoryWarning()** gets called.

You can override this method to release (set all pointers to it to nil) additional memory (anything big that is not currently in use and that can be easily re-created back again).

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

    //your code here
}
```

Segue

(UIStoryboardSegue)

- connects view controllers
- provides a mechanism to pass a context through view controllers

`UIStoryboardSegue has these properties:`

Source

```
var source: UIViewController
//a controller from which the transition is done
```

Destination

```
var destination: UIViewController
// a controller to which the transition is done
```

Identifier

```
var identifier: String?
// used to distinguish between different transitions which are done
```

Creating & performing segues

There are several ways to create and perform segues

- Create and perform in Interface Builder (IB)

(e.g. ctrl+drag from UIButton will work only by pressing UIButton)

- Create in IB, perform in code
- Create in code and perform in code (we won’t cover this)

To perform a segue from the code you call a method

```
func performSegue(withIdentifier identifier: String,
sender: Any?)
```

where **identifier** defines which segue to perform. If you create a segue through IB, a **sender** is the instigating object from a storyboard (e.g. a button)

UIViewController + UIStoryboardSegue

When a segue happens, you get a callback just before the visual transition occurs:

```
func prepare(for segue: UIStoryboardSegue, sender: Any?)
```

Note that when the segue happens, it always creates **a new instance** of view controller. Use this callback to pass a context (any needed data) to a view controller that is about to be displayed. BUT note that the outlets of the destination view controller are not set yet *(remember viewDidLoad() method?)*, so keep this in mind when you set up the destination controller.

Block a segue

You can prevent a segue from performing (works only for segues that are being performed *not through the* code)

```
func shouldPerformSegue(withIdentifier identifier:
String, sender: Any?) -> Bool
```

Unwind Segue

Is used to dismiss a top view controller and return to some existing view controller (e.g. when user presses Cancel or Done button).
UIKit will walk through all presented view controllers to find the first one that implements a special method, and drop all top VCs before.

To create an unwind segue:

1. Implement a method with a special asignature in a VC which you want to return into

```
@IBAction func backToMainWC(segue: UIStoryboardSegue) {
}
```

2. Ctrl+drag to **Exit** in VC you want to exit from and choose a signature created on first step
3. Use as a regular segue