iPhone Programming
Controls, Buttons, and Alerts

Tim Gegg-Harrison

Enumerations

- An **enumeration** defines a common type for a group of related values and enables you to work with those values in a type-safe way within your code.

- Enumerations in Swift are first-class types in their own right.

```
enum SomeEnumeration {
    // enumeration definition goes here
}
```

Enumerations …

```
enum CompassPoint {
    case north
    case south
    case east
    case west
}

var directionToHead: CompassPoint
   = CompassPoint.west

var previousDirection: CompassPoint
   = .south
```

Enumerations …

```
switch directionToHead {
case .north:
    println("Lots of planets have a north")
case .south:
    println("Watch out for penguins")
case .east:
    println("Where the sun rises")
case .west:
    println("Where the skies are blue")
}
```

Controls

- Graphical objects used by the user to interact with the application
- **`UIControl`**
  - **`isEnabled`**
  - **`isHighlighted`**
  - **`isSelected`**
  - **`state`**
- **`UIControl`** and its subclasses use the target-action mechanism (Swift's event handling mechanism) to handle changes to the control

Controls …

- **`UIControl`** attributes
  - **`isEnabled`**
    - Boolean attribute that represents whether the control is enabled or not
    - If the value is **`false`** then the user's touch events are ignored
    - Default is **`true`**
  - **`isHighlighted`**
    - Boolean attribute that controls whether the control is highlighted or not
    - When the user touches the control, the value changes to **`true`** and the control is highlighted
    - When the user leaves the control, the value changes to **`false`** and the control is not highlighted

Controls …

- `UIControl` attributes …
  - **`isSelected`**
    - Boolean attribute that indicates whether the control is selected or not
    - Most subclasses of `UIControl` do not use this attribute
    - `UISwitch` uses this attribute
  - **`state`**
    - Read-only attribute that defines the state of the control
    - `UIControlState`
      - `UIControlState.highlighted`
      - `UIControlState.disabled`
      - `UIControlState.normal`

Controls …

- The `ControlEvents` is a bit-mask specifying the control events that trigger the sending of an action message to the target:
  - **`UIControlEvents.valueChanged`**
    - value of the control has changed (e.g., slider moved)
  - **`UIControlEvents.editingDidBegin`**
    - control started editing (e.g., within a text field)
  - **`UIControlEvents.editingDidEnd`**
    - touch ending the editing of a field by leaving its bounds
  - **`UIControlEvents.touchDown`**
    - single tap touch-down inside the control’s bounds
  - **`UIControlEvents.touchUpInside`**
    - single tap touch-up inside the control’s bounds

Controls …

- Target-action methods available in `UIControl`:

func addTarget(Any?, action: Selector, for: UIControlEvents)
Associates a target object and action method with the control.

func removeTarget(Any?, action: Selector?, for: UIControlEvents)
Stops the delivery of events to the specified target object.

func actions(forTarget: Any?, forControlEvent: UIControlEvents)
Returns the actions performed on a target object when the specified event occurs.

var allControlEvents: UIControlEvents
Returns the events for which the control has associated actions.

var allTargets: Set<AnyHashable>
Returns all target objects associated with the control.

Buttons

- The `UIButton` class is a control that encapsulates the behavior of buttons.

- Some of the available button types:

case custom
No button style.
case system
A system style button, such as those shown in navigation bars and toolbars.
case detailDisclosure
A detail disclosure button.
case infoLight
An information button that has a light background.
case infoDark
An information button that has a dark background.
case contactAdd
A contact add button.
static var roundedRect: UIButtonType
A rounded-rectangle style button.

Buttons …

- Creating a button:

```
let button: UIButton

button = UIButton(type: UIButtonType.custom)
button.frame = CGRect(x: centerX-32, y:
  centerY-32, width: 64, height: 64)
button.setImage(#imageLiteral(resourceName:
  "play.png"), for: UIControlState.normal)
button.addTarget(self, action:
  #selector(ViewController.buttonPressed), for:
  UIControlEvents.touchUpInside)

self.view.addSubview(button)
```

Buttons …

- Implementation of the `buttonClicked` method:

```
@objc func buttonClicked() {


}
```

Closures

- Closure
  - Encapsulates a piece of code and a binding to local variables
  - Anonymous method with a return type and parameters:

```
{ (parameters) -> return_type in
    statements
}
```

Closures

- Passing a closure to a method is similar to passing any other object to a method:

```
func someFunctionThatTakesAClosure(closure:
  () -> ()) {
    // function body goes here
}
```

- Invoking the function with a closure:

```
someFunctionThatTakesAClosure({() -> () in
    // closure's body goes here
}
```

Alerts

- Displays alert messages to user
- Prior to iOS 8, alerts were handled with UIAlertView and UIAlertViewDelegate (alertView method)
- Current way to implement alerts:
  - UIAlertController
    - addAction
    - presentViewController
  - UIAlertAction
    - title
    - style

Alerts …

```
let alert: UIAlertController
    = UIAlertController(title: "title text", message:
        "message text", preferredStyle:
        UIAlertControllerStyle.alert)

alert.addAction(UIAlertAction(title: "Yes", style:
    UIAlertActionStyle.default, handler:
    {(action: UIAlertAction!) -> Void in
        // code if user presses “Yes” button
}))
alert.addAction(UIAlertAction(title: "No", style:
    UIAlertActionStyle.default, handler:
    {(action: UIAlertAction!) -> Void in
        // code if user presses “No” button
}))

self.present(alert, animated: true, completion:
    {() -> Void in
        // code when alert controller is presented
})
```