UnitConverter
Lesson 8

Description

Extract the responsibilities of the `UIPickerViewDataSource` protocol from the controller into the `TemperatureRange` view model.

Learning Outcomes

- Construct a model by implementing a class definition.
- Reorganize code in model and controller implementations to establish a separation of concerns.
- Practice adopting protocols and implementing class methods to conform to a protocol.
- Practice creating outlet connections to controller properties.

Vocabulary

`import`	inheritance	protocol adoption
`NSObject`	`UIPickerViewDataSource`	outlet connection
`@IBOutlet`	property	

Materials

- **UnitConverter Lesson 8** Xcode project

Opening

What changes do we need to make to the controller and our new view model to satisfy the temperature picker?

Agenda

- Discuss the changes necessary for enabling the `TemperatureRange` view model to act as the picker view's `dataSource`.
- Change the `TemperatureRange` class `import` statement to provide access to the `UIPickerViewDataSource` type.

```
import UIKit
```

- Update the `TemperatureRange` class to inherit from `NSObject` and to adopt the `UIPickerViewDataSource` protocol.

```
class TemperatureRange: NSObject, UIPickerViewDataSource {
```

- Explain why the inheritance hierarchy of the `UIPickerViewDataSource` protocol motivates the need to inherit from `NSObject`.
- Using the Xcode Documentation and API Reference (⇧⌘0), explore the `NSObject` class.
- Remove the `UIPickerViewDataSource` protocol adoption from the `ViewController` class definition.

```
class ViewController: UIViewController, UIPickerViewDelegate {
```

- Move the `temperatureValues` property out of the controller and into the `TemperatureRange` class. Remove the `private` access control modifier, and shorten its name to `values`.

```
let values = (-100...100).map { $0 }
```

- Explain why the property is no longer private, because code in other files will need to access the `values` property.
- Discuss the semantic redundancy of a `TemperatureRange temperatureValues` property and the decision to shorten the property name.
- Move the controller methods `numberOfComponentsInPickerView:` and `pickerView:numberOfRowsInComponent:` into the `TemperatureRange` class, and replace the reference to `temperatureValues` with `values`.

```
func numberOfComponentsInPickerView(pickerView: UIPickerView) -> Int {
   return 1
}

func pickerView(pickerView: UIPickerView,
   numberOfRowsInComponent component: Int) -> Int {
   return values.count
}
```

- View the `ViewController` class, and observe the red error indicators.
- Discuss the causes of the red error indicators in the `ViewController` class and plan the remaining controller changes.
- Discuss how the controller methods `pickerView:titleForRow:forComponent:` and `pickerView:didSelectRow:inComponent:` also need access to the view model.
- Using Interface Builder and the Assistant Editor (⌥⌘↩), Control-drag an outlet connection from the TemperatureRange object to the controller class, to create a new property.

```
@IBOutlet var temperatureRange: TemperatureRange!
```

- Update the controller methods `pickerView:titleForRow:forComponent:` and `pickerView:didSelectRow:inComponent:` to use the new `temperatureRange` property, replacing references to `temperatureValues` with `temperatureRange.values`.

```
func pickerView(pickerView: UIPickerView, titleForRow row: Int,
   forComponent component: Int) -> String! {
   let celsiusValue = temperatureRange.values[row]
   return "\(celsiusValue)°C"
}

func pickerView(pickerView: UIPickerView, didSelectRow row: Int,
   inComponent component: Int) {
   let degreesCelsius = temperatureRange.values[row]
   temperatureLabel.text =
      "\(converter.degreesFahrenheit(degreesCelsius))°F"
}
```

- Run the app (⌘R), select a temperature, and observe the converted value.
- Discuss the remaining controller code, and how it only manages communication between the view and the models, and updates the view.

Closing

What about the user experience of our app? What problems do you see?

Modifications and Extensions

- Extract the remaining `UIPickerViewDelegate` methods into another view model object, and refactor the controller to use both the separate delegate and data source. Analyze the benefits and drawbacks to this approach.

Resources

The Swift Programming Language: Protocols https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/Protocols.html

UIKit User Interface Catalog: Picker Views https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/UIKitUICatalog/UIPickerView.html

UIPickerView Class Reference https://developer.apple.com/library/ios/documentation/UIKit/Reference/UIPickerView_Class/

UIPickerViewDataSource Protocol Reference https://developer.apple.com/library/ios/documentation/iPhone/Reference/UIPickerViewDataSource_Protocol/index.html

Delegates and Data Sources http://developer.apple.com/library/ios/documentation/general/conceptual/CocoaEncyclopedia/DelegatesandDataSources/DelegatesandDataSources.html

Cocoa Core Competencies: Model Object https://developer.apple.com/library/ios/documentation/General/Conceptual/DevPedia-CocoaCore/ModelObject.html

The Swift Programming Language: Classes and Structures https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/ClassesAndStructures.html

Xcode Overview: Connecting User Interface Objects to Code https://developer.apple.com/library/ios/documentation/ToolsLanguages/Conceptual/Xcode_Overview/edit_user_interface.html#//apple_ref/doc/uid/TP40010215-CH6-SW3

Interface Builder Connections Help: Creating an Outlet Connection https://developer.apple.com/library/ios/recipes/xcode_help-IB_connections/chapters/CreatingOutlet.html