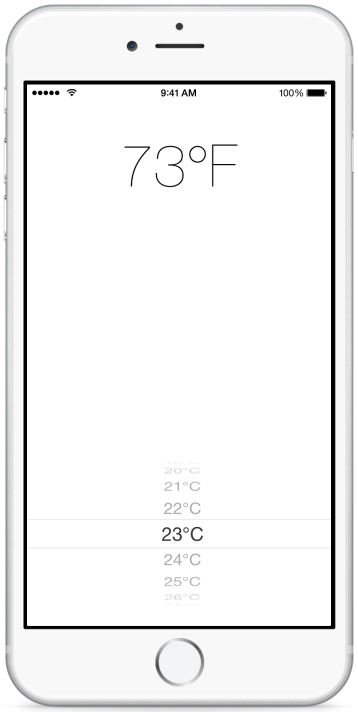
UnitConverter
Lesson 6

Description

Extract the controller's unit conversion code into a `UnitConverter` model.



Learning Outcomes

- Describe the model-view-controller pattern and distinguish the responsibilities of the model and controller.
- Construct a model by creating a Swift class.
- Practice declaring properties and writing method definitions.
- Reorganize code in model and controller implementations.
- Integrate a model, view and controller to complete an app feature.

Vocabulary

Model-View-Controller	model	separation of concerns
encapsulation	class definition	property
access control modifier	private	method
parameter	return type	string interpolation

Materials

- **UnitConverter Lesson 6** Xcode project
- **Model-View-Controller** presentation

Opening

What should the responsibilities of the controller be?

Agenda

- Discuss the existing controller code, and indicate how it seems to be concerned with displaying temperature values and converting temperature.
- Present the concept of Model-View-Controller.
- Discuss how models are Swift classes that encapsulate something in the problem domain, such as unit conversion.
- Add a new Swift class (⌘**N**) to the project for a `UnitConverter` model.

```
import Foundation

class UnitConverter {

}
```

- Discuss why the temperature conversion code in the controller `pickerView:didSelectRow:inComponent:` method belongs in the model.
- Add a `degreesFahrenheit:` method to the `UnitConverter` class.

```
func degreesFahrenheit(degreesCelsius: Int) -> Int {
   return Int(1.8 * Float(degreesCelsius) + 32.0)
}
```

- Explain the components of the method signature, including the method name, return type, parameter name and parameter type.
- In the `ViewController` class, declare a new `private` property for a `UnitConverter` object.

```
private let converter = UnitConverter()
```

- Discuss the choice to declare the property `private` and how the default property value is assigned during controller initialization.
- Update the `pickerView:didSelectRow:inComponent:` method to use the `UnitConverter` `degreesFahrenheit:` method.

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

- Discuss how the `degreesFahrenheit:` method call is used within the string interpolation.
- Run the app (⌘R), select a temperature, and observe the converted value.
- Discuss how the model now encapsulates the temperature conversion; and how the controller is now concerned with obtaining a temperature value from the picker view, obtaining a converted temperature from the model, and updating the view.

Closing

Should the controller be concerned with being the picker view `dataSource`? Should the `UnitConverter` model handle this? Why or why not?

Modifications and Extensions

- Refactor the body of `pickerView:didSelectRow:inComponent:` into a single statement. Consider long lines of code and decide whether or not the code has become more or less readable.
- Add a `degreesCelsius` property to the `UnitConverter`, and replace the `degreesFahrenheit:` method with a computed property.

Resources

Start Developing iOS Apps Today: Using Design Patterns https://developer.apple.com/library/ios/referencelibrary/GettingStarted/RoadMapiOS/DesignPatterns.html

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

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

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

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

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

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