TP 6 - Flashcard

Adrien Humilière 30/03/2017

Part 1

- Create a new – Single View Application – project.
- Using Interface Builder, add a Label and change its name to **Flashcard Term**. Add layout constraints to the label to set its vertical position and to center it within the view.
- The user will navigate between the front and back of different flashcards. iOS provides navigation controllers to automatically manage transitions between multiple view controllers.
- Add a Navigation Controller to the storyboard. Interface Builder adds both a navigation controller and a table view controller. Delete the table view controller from the storyboard.
- Arrange the Navigation Controller to the left of the existing View Controller. Move the incoming arrow on the main View Controller to the Navigation Controller, to indicate that it is the initial view controller for the app. Control drag from the Navigation Controller to the View Controller, and select the `rootViewController` Relationship Segue.
- The navigation controller acts as a container that manages navigation between different view controllers, such as the flashcard term view controller and the yet-to-be-created definition view controller. Interface Builder automatically adds a navigation bar to the top of any view controllers that the navigation controller manages.
- Select the Flashcard Term label, and use the Editor » Resolve Auto Layout Issues » Update Frames menu item to adjust the constraint issue caused by the navigation bar.
- Select the navigation bar at the top of the View Controller, and set the title to *Term*.
- Rename the View Controller to Term Controller.
- Drag a new Bar Button Item to the navigation bar in the Term Controller, and change the button title to Definition.
- Run the app, and observe how the Term Controller view appears with a navigation bar and button at the top.
- Using Interface Builder, add another View Controller to the storyboard, placing it to the right of the Term Controller.
- Rename the new View Controller to Definition Controller.
- Drag a Text View, for holding lots of text, onto the Definition Controller interface. Add layout constraints by Control-dragging from the text view on the canvas to the View in

the Document Outline. Create constraints for the leading, trailing, top and bottom space relative to the View.

- Control-drag from the Definition button to the Definition Controller, select the show segue, and observe how Interface Builder represents the new relationship with an arrow between the two view controllers. Segues represent transitions from one view controller to another.
- Update the constraints to align with the navigation bar.
- Drag a Navigation Item onto the view and set the title to *Definition*.
- Run the app, observe the term appear, tap the Definition button, and observe the definition appear. Tap the Term button and observe the transition back to the term view. The navigation controller automatically manages the back button.

Part 2

- We now need a `Flashcard` model, to encapsulate a term and its definition. Add a new `Flashcard` class to the project.
- Declare `String` properties in the `Flashcard` class for a `term` and `definition`. Observe the error notice in the Xcode editor, that indicates the need for an initializer. Implement a parameterless initializer in the `Flashcard` class, with default values for `term` and `definition`.
- Add a parameterized initializer to the `Flashcard` class, taking a term and its definition in parameters.

```
1 init(term: String, definition: String) {
2     self.term = term
3     self.definition = definition
4 }
```

- The two initializers contain duplicate code that assigns initial values to each property. Update the parameterless initializer to invoke the parameterized initializer.
- Observe the error notice in the Xcode editor, indicating the need to declare the parameterless initializer as a convenience initializer. Update the parameterless initializer, declaring it as a convenience initializer.
- The interface needs to display a Flashcard term in the main view of the Term Controller scene.
- Using Interface Builder, create an outlet for the term label as a `ViewController` property.
- Update the label in `viewDidLoad` to use a default `Flashcard` object term.
- Run the app, and observe the label text.

Part 3

- We now need for a `Deck` model, representing a collection of `Flashcard` objects. Add a new `Deck` class to the project.
- The Deck model will manage a collection of `Flashcard` objects, but the controller will use methods to "ask" a `Deck` for a card, rather than accessing the collection of `Flashcard` objects directly. Add a `private [Flashcard]` property to the `Deck` class.

```
1 private var cards = [Flashcard]()
```

- The cards property is `private` to hide how the `Deck` class manages the collection of `Flashcard` objects. Initializing a `Deck` should fill the cards array with a collection of `Flashcard` objects.
- Implement the `Deck` initializer, using a dictionary of term-definition pairs for `Flashcard` objects.

```
1 init() {
2     let cardData = [
3         "controller outlet" : "A controller view property, ↩
    marked with IBOutlet.",
4         "controller action": "A controller method, marked ↩
    with IBAction, that is triggered by an interface event."
5     ]
6     for (term, definition) in cardData {
7         cards.append(Flashcard(term: term, definition: ↩
    definition))
8     }
9 }
```

- The initializer is transforming an array of flashcard data into an array of Flashcard objects. It is a good opportunity for using `map` method. Replace the for-in loop with a verbose call of map.

```
1 cards = cardData.map(
2     { (term: String, definition: String) -> Flashcard in
3         return Flashcard(term: term, definition: definition)
4     })
```

- The `map` function is passed a closure expression; it invokes the closure for each key-value pair in the dictionary, builds an array with each returned `Flashcard` object, and assigns

the resulting array to the cards property. Swift can infer the type of the closure expression from the data type of the `cardData` dictionary and the cards array. Refactor the `map` call, removing the explicit type annotations.

```
cards = cardData.map( { term, definition in
    return Flashcard(term: term, definition: definition)
})
```

- Because the closure expression only contains one statement, Swift also infers an implicit `return`. Refactor the `map` call, removing the explicit `return`.

```
cards = cardData.map( { term, definition in
    Flashcard(term: term, definition: definition)
})
```

- Because the closure expression is the last argument to `map`, we can use the Swift trailing closure expression syntax; Swift provides shorthand argument names, removing the need for the explicit term and definition arguments. Refactor the `map` call, using a trailing closure expression and shorthand argument names.

```
cards = cardData.map { Flashcard(term: $0, definition: $1) }
```

- Because the initializer no longer appends `Flashcard` objects to the mutable cards array property, the property can now be constant. Modify the `cards` property declaration to a constant, without a default value. The `cards` property declaration no longer instantiates an empty `[Flashcard]` array, since the initializer uses `map` to assign the property its `[Flashcard]` value.

Part 4

- Using Interface Builder and the Document Outline, select the Term Controller and use the Identity Inspector to reveal the binding to the custom ViewController class. Each individual view controller in the storyboard can be associated with a specific class within the project.
- We have a naming inconsistency of Term Controller in the storyboard, and the `ViewController` class name. Using the Project Navigator, rename `ViewController.swift` to `TermController.swift`, and update the class name to `TermController`.
- In Interface Builder, select the Term Controller and use the Identity Inspector to change the Custom Class to `TermController`.

- Add a Deck property to the TermController class, we a default `Deck` object.
- The `TermController` `viewDidLoad` method will draw a random `Flashcard` from the deck, and use that `Flashcard` term property to update the text label.
- Add a naive randomCard method to the `Deck` class.

```
func randomCard() -> Flashcard {
    let randomIndex = Int(arc4random_uniform(UInt32(cards.count)))
    return cards[randomIndex]
}
```

- The `randomCard` method should not return a `Flashcard` object when the deck is empty, and `cards.count` is 0. Improve the `randomCard` method with an optional return type.

```
func randomCard() -> Flashcard? {
    if cards.isEmpty {
        return nil
    } else {
        let randomIndex = Int(arc4random_uniform(UInt32(cards.count)))
        return cards[randomIndex]
    }
}
```

- The `randomCard` method has no parameters, only does the necessary work to return a value, and "feels" like a property of a `Deck`. Replace the `randomCard` method with a computed property.

```
var randomCard: Flashcard? {
    if cards.isEmpty {
        return nil
    } else {
        return cards[Int(arc4random_uniform(UInt32(cards.count)))]
    }
}
```

- In `TermController`, update the implementation of `viewDidLoad` to draw a `randomCard`, and use that card to update the text label.

```
override func viewDidLoad() {
    super.viewDidLoad()
    if let flashcard = deck.randomCard {
        termLabel.text = flashcard.term
    }
}
```

- Run the app and observe the random card term on the screen. Tap the Definition button, observe how the default text view text appears, and navigate back to the first view controller.

Part 5

- Add a new Swift class to the project called `DefinitionController` extending `UIViewController`.
- Using Interface Builder, select the Definition Controller and use the Identity Inspector to set the Class to `DefinitionController`.
- Run the app, tap the Definition button, and observe how the default text view text still appears.
- `TermController` obtains a `Flashcard` object. We need to provide the same `Flashcard` object to the `DefinitionController`, so that it can display the definition of the particular `Flashcard`.
- Add a `Flashcard?` property to the `DefinitionController` class. The property is optional, because the `DefinitionController` initializer will not initialize the property; the property is a variable, because the controller will present definitions of different `Flashcard` objects.
- Using Interface Builder, select the Definition Controller and create a connection from the text view to an outlet in the `DefinitionController` class.
- Implement a `viewDidLoad` method in the `DefinitionController`, to set the definition text using the `Flashcard` property.
- In the documentation, examine the `UIViewController` method `prepareForSegue:sender:`. Before a segue is performed, the `prepareForSegue:sender:` method is called, and receives a reference to both a `UIStoryboardSegue` object and a reference to the interface control that triggered the segue. `UIStoryboardSegue` has two properties `sourceViewController` and `destinationViewController` that will be useful.
- Add a `Flashcard?` property to the `TermController` class.
- Update the `TermController` `viewDidLoad` implementation, to assign a value to the `Flashcard` property and use it after that.

- Implement `prepareForSegue:sender:` in the `TermController` class.

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    if let definitionController =
        segue.destinationViewController as? DefinitionController {
        definitionController.flashcard = flashcard
    }
}
```

- An object is retrieved from the segue, is casted to a `DefinitionController` using the `as?` type cast operator, and the `TermController` uses its `flashcard` property to assign a `Flashcard` object to the `DefinitionController` `flashcard` property.
- Run the app, tap the Definition button, and observe the correct definition appear.

Part 6

- We want a new random term to be displayed every time the `TermController` is presented.
- Explore the `UIViewController` documentation.
- Find the best method to implement to make sure the `TermController` is updated each time it appears. Implement it.
- Run the app, tap the Definition button, observe the corresponding definition, tap the Term (back) button, and observe a new (likely, due to the random Flashcard selection) term appear. Move back and forth between term and definition to observe the changes.