Глава 10. Передача данных между контроллерами

В этой главе вы:

- узнаете основные способы передачи данных между вью контроллерами в приложении.

Приложение – это множество взаимосвязанных между собой экранов. Одни экраны просто сменяют друг друга, а другие подразумевают односторонний или двусторонний обмен данными. Вы уже довольно неплохо разбираетесь в вопросах навигации между сценами – segue и Navigation Controller для вас уже не темные лошадки. Но пока что сцены всех разработанных вами проектов функционировали независимо друг от друга, а это значит, что их совершенно ничего не объединяло. Однако, чем глубже вы будете погружаться в iOS-разработку, тем острее будет вставать вопрос взаимодействия отдельных сцен посредством обмена данными.

В качестве примера рассмотрим страницу профиля в медицинском приложении «**Голдлайн**». На рисунке 10.1 показан процесс выбора диагноза путем перехода между сценами. Изначально диагноз не выбран, но при нажатии на соответствующую ячейку происходит переход к экрану выбора, а после выбора измененные данные отображаются на экране профиля.

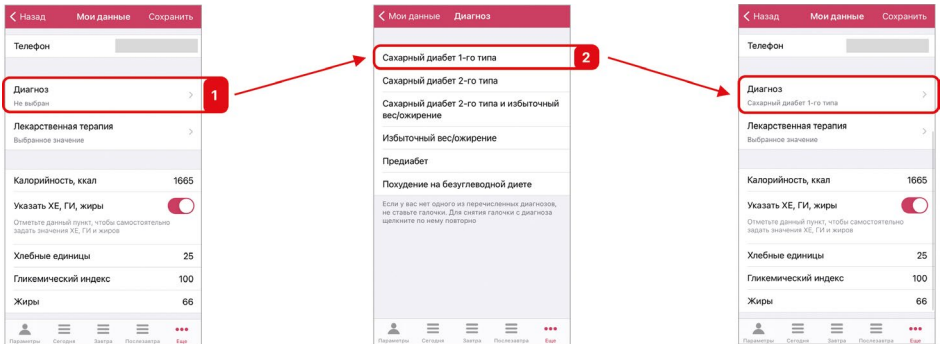

Рис. 10.1. Передача выбранного диагноза между экранами

Какое бы приложение вы не разрабатывали (простое или сложное), перед вами обязательно встанет вопрос реализации обмена данными между вью контроллерами. В этой главе мы рассмотрим некоторые из доступных в Swift и Xcode способов решения этой задачи.

10.1 Создание проекта

На рисунке 10.2 показана типовая схема взаимодействия контроллеров внутри приложения.

Рис. 10.2. Схема взаимодействия контроллеров в приложении

Контроллер А отображает некоторые данные. При необходимости их изменения будет происходить переход к контроллеру Б. После изменения обновленные данные будут вновь передаваться в контроллер А для отображения.

Руководствуясь приведенной схемой, создадим проект, на основе которого и будем изучать вопрос обмена данными между контроллерами. В составе проекта будут созданы две сцены, а для навигации между ними воспользуемся навигационным контроллером. На первом экране (контроллер А) будет размещена текстовая метка для отображения, а на втором (контроллер Б) – текстовое поле для изменения. Значения в текстовой метке и текстовом поле всегда должны совпадать. При переходе с контроллера А будут передаваться данные для заполнения текстового поля, а при его изменении и сохранении – обратно, для вывода в текстовой метке.

В этой главе мы рассмотрим несколько способов передачи данных между сценами. Для каждого из них будем создавать свой набор кнопок на сцене, а также порассуждаем об особенностях каждого подхода.

Начнем с создания нового проекта.

- Создайте проект **TransferApp**. В качестве шаблона выберите **App**.
- Откройте файл **Main.storyboard**.

В составе проекта уже есть один вью контроллер. Будем называть его **контроллер А** или **сцена А**.

В качестве базы для навигации между сценами будем использовать Navigation Controller.

- Оберните контроллер А в Navigation Controller (пункт меню **Editor > Embed In > Navigation Controller**).

Изменим внешний вид сцены.

- Измените стиль оформления заголовка в Navigation Bar на **Large**:
  - в Navigation Controller на панели **Attributes Inspector** активируйте пункт **Prefers Large Titles**;
  - в Navigation Item контроллера А на панели **Attributes Inspector** измените значение поля **Large Title** на **Always**.
- Измените заголовок контроллера А, выводимый в Navigation Bar, на «**Сцена А**».
- Измените фоновый цвет сцены А на фиолетовый.

Далее на сцене разместим и настроим текстовую метку (Label), в которой будет выводиться актуальное строковое значение.

- Разместите в центре сцены А текстовую метку. Для позиционирования самостоятельно создайте необходимые констрейнты.

> **Примечание** Даже если вы не сможете создать необходимые ограничения для размещаемых на сцене элементов, это никак не повлияет на функциональную составляющую проекта. Тем не менее, я настоятельно рекомендую вам постараться и (при возникновении сложностей) используя уже изученный ранее материал, а также помощь в нашем чате в Telegram, создать требуемые ограничения.

- Измените цвет текста метки на белый, а размер шрифта на 30.
- Свяжите сцену А с классом **ViewController**.
- В классе **ViewController** создайте аутлет **dataLabel** и свяжите его с текстовой меткой на сцене А (листинг 10.1).

ЛИСТИНГ 10.1

```
class ViewController: UIViewController {
    @IBOutlet var dataLabel: UILabel!
    // ...
}
```

На этом подготовка контроллера А завершена (рис. 10.3).

Теперь добавим на storyboard дополнительный контроллер (контроллер Б), отвечающий за смену строкового значения.

Рис. 10.3. Storyboard проекта

- Разместите на сториборде новый вью контроллер.
- Измените фоновый цвет сцены на зеленый.
- Измените заголовок сцены, выводимый в Navigation Bar, на «**Сцена Б**». Для этого потребуется добавить на сцену Navigation Item.
- Разместите в центре сцены Б текстовое поле (Text Field). Для позиционирования используйте ограничения.
- Определите отступы слева и справа от текстового поля в 30 точек.
- Выровняйте содержимое внутри текстового поля по центру.

Визуальная составляющая сцены готова (рис. 10.4). Обратите внимание, что пока мы не создадим segue между первым и вторым контроллерами, Navigation Bar не будет отображаться на storyboard.

Теперь создадим класс, который будет управлять сценой Б.

- Создайте новый файл **SecondViewController.swift** с классом **SecondView Controller**, который является потомком **UIViewController** (листинг 10.2).

ЛИСТИНГ 10.2

```
class SecondViewController: UIViewController {
    // ...
}
```

- Свяжите контроллер Б с классом **SecondViewController**.
- В классе **SecondViewController** создайте аутлет **dataTextField** и свяжите его с текстовым полем на сцене Б (листинг 10.3).

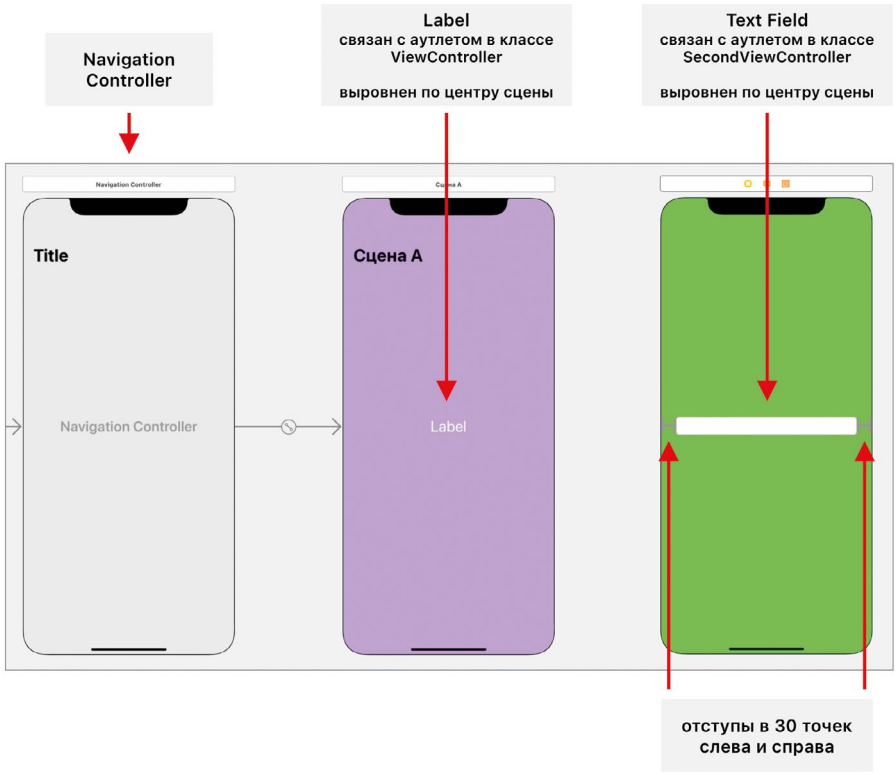

Рис. 10.4. Storyboard проекта

ЛИСТИНГ 10.3

```
class SecondViewController: UIViewController {
    @IBOutlet var dataTextField: UITextField!
    // ...
}
```

Теперь укажем Storyboard ID для каждого из контроллеров на сториборде.

- Для контроллера А в поле **Storyboard ID** укажите **ViewController**.
- Для контроллера Б в поле **Storyboard ID** укажите **SecondViewController**.

> **Примечание** Не перепутайте Storyboard ID для каждого из контроллеров, он идентичен имени класса, связанного со сценой.

Весь дальнейший материал, описываемый в главе, будет основан на использовании данного проекта. В последующих разделах мы рассмотрим несколько способов передачи данных между контроллерами в каждую из сторон.

10.2 Передача данных от А к Б с помощью свойств

Наиболее простым способом передать данные из одного контроллера в другой является использование свойств. Всякий раз при создании экземпляра вью контроллера, к которому производится переход, данные инициализируются в предназначенное для них свойство в этом экземпляре. При выводе сцены на экран данные в этом свойстве могут быть использованы для наполнения графических элементов (например, для заполнения текстового поля).

> **Примечание** Прошу обратить внимание, что в этой главе мы рассматриваем вопросы передачи данных между контроллерами, а не последующего использования этих данных для обновления элементов интерфейса.

- На сцене А под текстовой меткой разместите кнопку «**Изменить с помощью свойства**». С помощью ограничений укажите отступ в 30 пикселей сверху (от текстовой метки), слева и справа.
- Измените фоновый цвет кнопки на синий, цвет шрифта на белый, а размер текста на 20 (рис. 10.5).

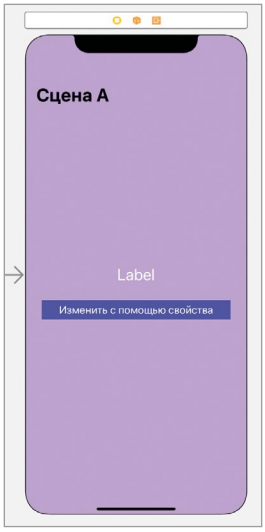

Рис. 10.5. Кнопка на сцене А

При нажатии созданной кнопки будет происходить передача данных в контроллер Б с последующим переходом к его сцене. Данные будут записываться в специальное свойство **updatingData**.

- В классе **SecondViewController** объявите свойство **updatingData** типа **String** (листинг 10.4).

ЛИСТИНГ 10.4

```
var updatingData: String = ""
```

Свойство **updatingData** будет использоваться для заполнения данными текстового поля на сцене Б. Каждый раз при отображении сцены значение текстового поля будет обновляться в соответствии со значением свойства.

► Дополните код класса **SecondViewController** методами из листинга 10.5.

ЛИСТИНГ 10.5

```
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    updateTextFieldData(withText: updatingData)
}

// обновляем данные в текстовом поле
private func updateTextFieldData(withText text: String) {
    dataTextField.text = text
}
```

Метод **viewWillAppear** относится к жизненному циклу вью контроллера. Напомню, он вызывается при каждом отображении сцены на экране, а не только при первом, как **viewDidLoad**.

Теперь реализуем непосредственно передачу данных и переход к сцене Б.

► В классе **ViewController** создайте экшн-метод **editDataWithProperty** (листинг 10.6).

ЛИСТИНГ 10.6

```
@IBAction func editDataWithProperty(_ sender: UIButton) {
    // получаем вью контроллер, в который происходит переход
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    let editScreen = storyboard.instantiateViewController(withIdentifier:
"SecondViewController") as! SecondViewController

    // передаем данные
    editScreen.updatingData = dataLabel.text ?? ""

    // переходим к следующему экрану
    self.navigationController?.pushViewController(editScreen, animated:
true)
}
```

► Свяжите вызов метода **editDataWithProperty** с нажатием кнопки «**Изменить с помощью свойства**» на сцене А.

► Запустите приложение.

При нажатии на кнопку «**Изменить с помощью свойства**» значение текстовой метки контроллера А передается в текстовое поле контроллера Б. Это очень простой, но наиболее очевидный и понятный способ передачи данных между контроллерами.

На что стоит обратить внимание

Работая над программным кодом ваших приложений, всегда старайтесь думать о том, а не навредит ли принятое решение архитектуре проекта. В частности, в использованной реализации создается излишняя связанность между вью контроллерами, так как внутри одного контроллера происходит непосредственная работа с другим контроллером. Если вдруг потребуется изменить название или тип свойства **updatingData**, или подставить вместо **SecondViewController** совершенно иной контроллер, это приведет к необходимости внесения изменений и в класс **ViewController**. Излишняя связанность не очень хорошо сказывается на архитектуре приложения, так как она приводит к необходимости порой неочевидных изменений.

Можно избежать подобной ситуации, создав между контроллерами прослойку в виде протокола, описывающего требования к наличию свойству **updatingData**. Подписав на протокол класс **SecondViewController** внутри метода **editDataWithProperty**, мы сможем производить тайпкастинг (приведение) не к типу **SecondViewController**, а к созданному протоколу:

```
protocol UpdatingDataController: class {
        var updatingData: String { get set }
}

class SecondViewController: UIViewController, UpdatingDataController {
        var updatingData: String = ""
        // ...
}

class ViewController: UIViewController {
        // ...
        @IBAction func editDataWithProperty (_ sender: UIButton) {
        // ...
        var editScreen = storyboard.instantiateViewController(withIdentifi
er: "SecondViewController") as! UpdatingDataController
        // ...
    }
}
```

Такой подход позволяет скрыть за протоколом внутреннюю реализацию и конкретный тип контроллера, а значит при необходимости использовать любой вью контроллер, соответствующий протоколу **UpdatingDataController**.

10.3 Передача данных от Б к А с помощью свойств

Данные успешно передаются от контроллера А в контроллер Б и отображаются в текстовом поле. Теперь рассмотрим вопрос обратной передачи измененных данных в контроллер А.

В первую очередь, мы реализуем обновление текстовой метки на сцене А при ее отображении на экране. То есть, каждый раз, когда сцена появляется на экране, текст в метке должен измениться на актуальный.

► Добавьте в класс **ViewController** код из листинга 10.7.

ЛИСТИНГ 10.7

```
var updatedData: String = "Test data"

override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)
    updateLabel(withText: updatedData)
}

// Обновляем данные в текстовой метке
private func updateLabel(withText text: String) {
    dataLabel.text = updatedData
}
```

Принцип тот же самый, что был ранее использован в классе **SecondViewController**.

Для передачи обновленного значения из контроллера Б в контроллер А нам необходимы два элемента: измененное значение и контроллер назначения.

> **Примечание** Контроллер назначения – это вью контроллер, в который осуществляется переход.

Измененное значение может быть получено с помощью свойства **dataTextField**. А вот для получения контроллера назначения (в нашем случае — контроллера А) есть несколько вариантов исполнения.

Вариант 1. Так как отображение сцен производится внутри навигационного контроллера, можно обратиться к навигационному стеку, получить ссылку на предыдущий контроллер и обновить значение свойства **updatedData**.

Вариант 2. В классе **SecondViewContoller** создать свойство, хранящее ссылку на вью контроллер. При переходе от сцены А к сцене Б инициализировать этому свойству ссылку на класс **ViewController** и с его помощью обновлять значение свойства **updatedData**.

Каждый из указанных вариантов имеет свои положительные и отрицательные стороны, о которых мы поговорим ниже. Мы же воспользуемся первым.

- На сцене Б ниже текстового поля разместите кнопку «**Сохранить с помощью свойства**». Укажите ограничение на отступ в 30 точек сверху, слева и справа.
- Оформите кнопку в соответствии с использованным ранее стилем на сцене А (измените фон, размер текста и цвет шрифта) (рис. 10.6).

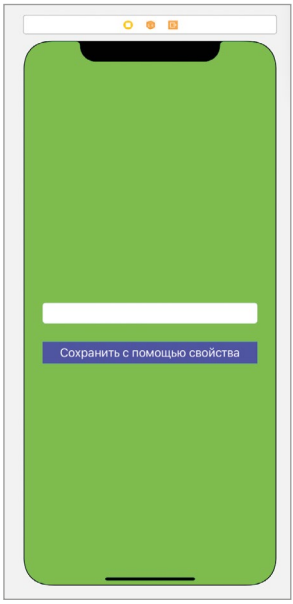

Рис. 10.6. Оформление сцены Б

- В классе **SecondViewController** реализуйте экшн-метод **saveDataWith Property** из листинга 10.8.

ЛИСТИНГ 10.8

```
@IBAction func saveDataWithProperty(_ sender: UIButton) {
    self.navigationController?.viewControllers.forEach { viewController in
        (viewController as? ViewController)?.updatedData = dataTextField.
text ?? ""
    }
}
```

- Свяжите вызов метода **saveDataWithProperty** с нажатием кнопки «**Сохранить с помощью свойства**».
- Запустите проект.

После перехода на сцену Б, изменения данных в текстовом поле и нажатия на кнопку сохранения данные в контроллере А обновляются, а по возвращению на сцену А мы видим уже обновленное значение.

> **Примечание** Вместо создания свойства **updatedData** можно было обновлять значение в текстовой метке вызовом метода **updateLabel** класса **ViewController**.

На что стоит обратить внимание

Выше я описал два варианта доступа к экземпляру класса **ViewController**: через навигационный стек и с помощью свойства. Хочу обратить ваше внимание, что при определенном уровне невнимательности использование свойства могло привести к утечке памяти! Если созданное свойство будет держать сильную ссылку на **ViewController**, контроллер Б не будет удален до тех пор, пока не будет удален контроллер А. В результате, возвращаясь со сцены Б к сцене А, экземпляр класса **SecondViewController** будет продолжать занимать ценную оперативную память.

> **Примечание** Если данный материал вызвал у вас затруднения, вернитесь к главе про управление памятью и ARC в первой книге.

Выходом из этой ситуации является хранение внутри **SecondViewController** слабой (**weak**) ссылки на экземпляр **ViewController** вместо сильной. Очень важно следить за тем, чтобы ссылающиеся друг на друга объекты при появлении соответствующих условий могли быть удалены из памяти средствами ARC.

Также вновь вернусь к вопросу сильной связанности классов, так как мы получили ту же самую проблему, что и ранее. И вновь решить ее можно с помощью протоколов. Достаточно создать протокол, требующий наличия свойства **updatedData**, подписать на него класс **ViewController** и использовать его при тайпкастинге вместо конкретного типа **ViewController**:

```
protocol UpdatableDataController: class {
        var updatedData: String { get set }
}

class ViewController: UIViewController, UpdatableDataController {
        var updatedData: String = ""
        // ...
}
class SecondViewController: UIViewController {
        // ...
        @IBAction func saveDataWithProperty(_ sender: UIButton) {
```

```
            self.navigationController?.viewControllers.forEach {
viewController in
                (viewController as? UpdatableDataController)?.updatedData =
dataTextField.text ?? ""
            }
        }
}
```

10.3 Передача данных от А к Б с помощью segue

Segue уже довольно хорошо вам известны. С их помощью в графическом интерфейсе **Interface Builder** можно с большим уровнем удобства решить вопрос навигации между сценами. При срабатывании segue пытается вызвать специальный метод **prepare** вью контроллера, из которого происходит переход. С его помощью можно выполнить требуемый код, в том числе, передать данные в контроллер назначения.

- На сцене А разместите новую кнопку «**Изменить с помощью segue**». Она должна находиться ниже на 30 точек уже существующей кнопки с отступами в 30 точек слева и справа. Не забывайте для этого использовать механизм ограничений.
- Измените стиль кнопки в соответствии со стилем других кнопок.
- Создайте segue от созданной кнопки к контроллеру Б. Для этого зажмите клавишу **Control** и перетяните кнопку на вторую сцену, а в выпадающем окне выберите «**Show**».

Сразу после создания на сцене Б отобразится Navigation Bar и заголовок сцены (рис. 10.7).

- На storyboard выделите созданный segue, откройте панель **Attributes Inspector** и в поле **Identifier** введите значение «**toEditScreen**».

Вью контроллер на storyboard может иметь множество переходов (segues). Если они используются только для презентации сцен, никаких сложностей не создается. Но если в рамках перехода также требуется выполнить какие-либо дополнительные действия (например, передать данные в контроллер назначения), необходимо использовать механизм, позволяющий отличить один segue от другого. И таким механизмом является идентификатор перехода (он был только что указан в поле **Identifier**).

Идентификатор – это уникальное, в пределах контроллера, строковое значение, позволяющее однозначно идентифицировать segue. При осуществлении

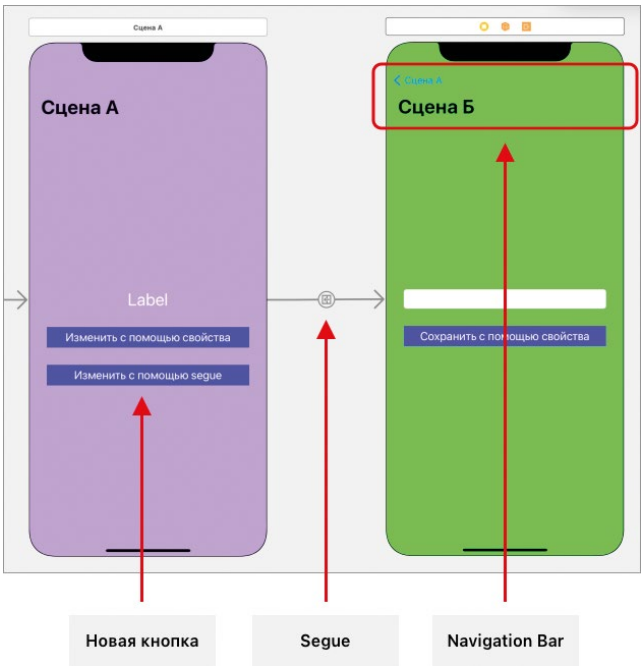

Рис. 10.7. Новые элементы на storyboard

перехода к новому контроллеру, используя идентификатор, можно определить какой именно segue сработал, и в зависимости от этого выполнить необходимые действия.

Для обработки перехода используется метод **prepare** класса **UIViewController**.

СИНТАКСИС

Метод `UIViewController.prepare(for:sender:)`
Вызывается в момент срабатывания segue непосредственно перед осуществлением перехода к новой сцене.

Аргументы

- `for: UIStoryboardSegue` – описывает сработавший segue. Используется для получения идентификатора, источника перехода (контроллера, с которого происходит переход) и контроллера назначения.
- `sender: Any?` – описывает элемент, вызвавший segue. Например, если segue привязан к нажатию кнопки, то в **sender** будет находиться экземпляр **UIButton**, соответствующий этой кнопке на сцене.

Особое внимание стоит обратить на аргумент **for** типа **UIStoryboardSegue**. С его помощью происходит идентификация segue, а также получение контроллеров источника и назначения.

СИНТАКСИС

Тип `UIStoryboardSegue`
Описывает segue (переход) между сценами.

Важные свойства

- `identifier: String` – идентификатор перехода.
- `destination: UIViewController` – контроллер назначения (к которому происходит переход).
- `source: UIViewController` – контроллер-источник, с которого происходит переход.

Реализуем метод **prepare**, позволяющий обработать переход и передать данные во вью контроллер.

► Добавьте в класс **ViewController** код из листинга 10.9.

ЛИСТИНГ 10.9

```
// Передача данных с помощью segue
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // определяем идентификатор segue
    switch segue.identifier {
    case "toEditScreen":
        // обрабатываем переход
        prepareEditScreen(segue)
    default:
        break
    }
}

// подготовка к переходу на экран редактирования
private func prepareEditScreen(_ segue: UIStoryboardSegue) {
    // безопасно извлекаем опциональное значение
    guard let destinationController = segue.destination as?
SecondViewController else {
        return
    }
    destinationController.updatingData = dataLabel.text ?? ""
}
```

► Запустите проект.

При нажатии на кнопку «**Изменить с помощью segue**» происходит переход к сцене редактирования, при этом значение из текстовой метки передается в текстовое поле. Сохранить данные (то есть передать измененные данные обратно) можно нажатием на созданную ранее кнопку «**Сохранить с помощью свойства**».

Огромным плюсом использования segue является их визуализация на storyboard. Стоит вам взглянуть на созданные связи между сценами, как становится ясно, каким образом происходит навигация внутри приложения, и как передаются данные. При этом реализация передачи данных, как вы только что убедились, очень проста.

10.4 Передача данных от Б к А с помощью unwind segue

Unwind segue – это особый вид перехода (segue), с помощью которого можно вернуться к одной из сцен, отображенных ранее. При этом у unwind segue есть несколько особенностей.

- Они не ограничены последней показанной сценой. Вы можете вернуться к любой сцене, показанной ранее, пропустив все промежуточные. Например, если в навигационном стеке четыре контроллера, вы с легкостью сможете создать unwind segue, который одним нажатием вернет вас к любому из этих контроллеров.
- Не имеет значения, каким образом выводится текущая сцена: с помощью метода **present** или внутри Navigation Controller – unwind segue способен произвести обратный переход.

Воспринимайте unwind segue, как обычный segue, но производящий переход в обратную сторону.

Для использования unwind segue необходимо реализовать метод с произвольным именем, принимающий всего один параметр типа **UIStoryboardSegue**. Самое важное, что данный **метод необходимо реализовывать в том контроллере, в который будет производиться обратный переход**, а не в том из которого производится возврат. То есть, если нам необходимо вернуться из контроллера **SecondViewController** к первой сцене, то метод реализуется в классе **ViewController** (а не **SecondViewController**).

► В классе **ViewController** реализуйте метод **unwind** из листинга 10.10.

ЛИСТИНГ 10.10

```
@IBAction func unwindToFirstScreen(_ segue: UIStoryboardSegue) {}
```

Для реализации unwind segue важен сам факт наличия данного метода, при этом нет необходимости наполнять его тело каким-либо кодом.

► На сцене Б разместите кнопку «**Сохранить с помощью unwind**». Она должна находиться ниже на 30 точек уже существующей кнопки с отступами в 30 точек слева и справа. Не забывайте для этого использовать механизм ограничений.

- Измените стиль кнопки в соответствии со стилем другим кнопок.

В результате на сцене появится новая кнопка (рис. 10.8).

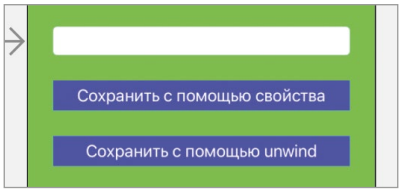

Рис. 10.8. Новая кнопка на сцене Б

- Зажмите клавишу **Control** и перетяните данную кнопку на элемент **Exit** в составе сцены. Найти его можно либо в **Document Outline**, либо прямо над сценой на сториборде (рис. 10.9).

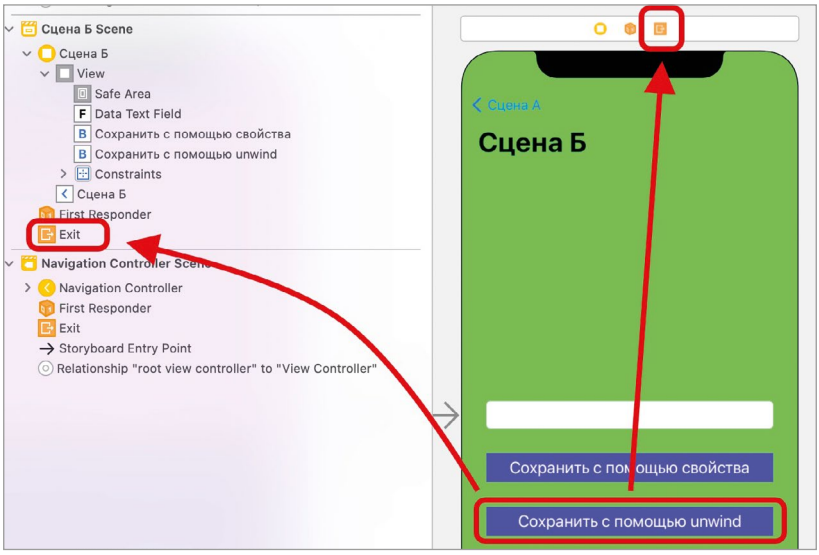

Рис. 10.9. Создание связи между кнопкой и элементом Exit

- Во всплывающем окне выберите пункт **unwindToFirstScreen** (рис. 10.10). Название пункта соответствует названию метода в первом контроллере.

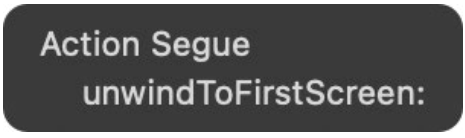

Рис. 10.10. Выбор unwind segue

Вы только что создали свой первый unwind segue. Обратите внимание, что в **Document Outline** в составе сцены Б появился новый элемент «**Unwind segue to unwindToFirstScreen**» (рис. 10.11).

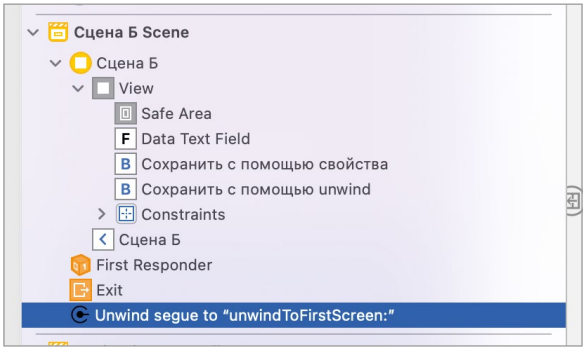

Рис. 10.11. Созданный unwind segue

- Запустите приложение.

При нажатии кнопки «**Сохранить с помощью unwind**» происходит переход к корневой сцене навигационного стека, а это говорит о том, что созданный unwind segue работает. Переход производится, но данные пока еще не передаются.

Реализуем обратную передачу данных. Для этого необходимо указать идентификатор unwind segue, и, используя его в методе **prepare**, передавать данные, как мы уже делали это в предыдущем разделе.

- В составе сцены Б в **Document Outline** выделите элемент «**Unwind segue to unwindToFirstScreen**».
- Откройте панель **Attributes Inspector**.
- В поле **Identifier** укажите «**toFirstScreen**».
- В классе **SecondViewController** реализуйте методы из листинга 10.11.

ЛИСТИНГ 10.11

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
    // определяем идентификатор segue
    switch segue.identifier {
    case "toFirstScreen":
        // обрабатываем переход
        prepareFirstScreen(segue)
    default:
        break
    }
}
```

```
// подготовка к переходу на первый экран
private func prepareFirstScreen(_ segue: UIStoryboardSegue) {
    // безопасно извлекаем опциональное значение
    guard let destinationController = segue.destination as? ViewController
else {
        return
    }
    destinationController.updatedData = dataTextField.text ?? ""
}
```

- Осуществите запуск приложения.

Теперь после нажатия кнопки «Сохранить с помощью unwind» происходит не только переход к корневой сцене, но и передача данных от контроллера Б контроллеру А. Сейчас у вас в запасе есть несколько способов передачи данных между контроллерами, и при этом они не конфликтуют между собой: вы можете осуществлять переходы и передачу данных любыми нажатиями на любые из кнопок в любых комбинациях.

10.5 Передача данных от Б к А с помощью делегирования

Вы уже хорошо знакомы с шаблоном «Делегирование». Напомню, его суть заключается в том, что объект делегирует (передает) ответственность за решение какой-либо задачи другому объекту. Данный подход прекрасно вписывается в контекст решения задачи передачи данных между контроллерами.

На рисунке 10.12 схематично показан принцип обмена данными на основе паттерна «Делегирование». Его суть заключается в том, что контроллер, изменяющий данные (контроллер Б), хранит ссылку на делегат (которым является контроллер А) и после изменения вызывает определенный метод делегата, передавая ему обновленные данные. А дальше, как использовать полученные данные, как с их помощью обновить интерфейс, как их сохранить – это проблема и задача делегата.

При этом наиболее верным подходом является скрытие делегата за протоколом, то есть когда контроллеры взаимодействуют не напрямую, а на основе протокола.

- В составе проекта создайте новый файл **DataUpdate.swift**.
- В созданном файле реализуйте протокол **DataUpdateProtocol** (листинг 10.12).

Рис. 10.12. Схема передачи данных

ЛИСТИНГ 10.12

```
protocol DataUpdateProtocol {
    func onDataUpdate(data: String)
}
```

Протокол **DataUpdateProtocol** содержит требование к наличию метода **onData Update**, который будет вызываться в контроллере А при изменении его данных в контроллере Б.

- Подпишите класс **ViewController** на протокол **DataUpdateProtocol** (листинг 10.13) и реализуйте метод **onDataUpdate**.

ЛИСТИНГ 10.13

```
class ViewController: UIViewController, DataUpdateProtocol {

    // ...

    func onDataUpdate(data: String) {
        updatedData = data
        updateLabel(withText: data)
    }
}
```

Ничего сложного в этом методе нет: при его срабатывании обновленное значение инициализируется свойству **updatedData** и обновляется содержимое текстовой метки.

Теперь необходимо, чтобы один класс стал делегатом другого. Хотя мы говорили об этом выше, как по вашему мнению – кто должен быть чьим делегатом и в каких вопросах? Так как обновление данных происходит на сцене Б, становится очевидно, что контроллер А должен быть делегатом контроллера

Б в вопросах обработки измененных данных. То есть, когда данные изменятся, контроллер Б вызовет метод **onDataUpdate** контроллера А, тем самым передавая ответственность.

- В классе **SecondViewController** объявите свойство **handleUpdated DataDelegate** (листинг 10.14).

ЛИСТИНГ 10.14

```
var handleUpdatedDataDelegate: DataUpdateProtocol?
```

> **Примечание** Довольно часто в учебных материалах для хранения делегата используют свойство с именем **delegate**. Но тут стоит заметить, что у контроллера может быть несколько делегатов, каждый из которых решает собственную задачу. По этой причине использование свойства с таким общим именем может привести к путанице.

- На сцене А создайте кнопку с текстом «**Изменить с помощью делегата**». Вновь, используя констрейнты, разместите ее в 30 точках ниже.
- Оформите кнопку в соответствии со стилем других кнопок.

При нажатии на кнопку «**Изменить с помощью делегата**» (рис. 10.13) контроллер, с которого происходит переход, будет устанавливаться в качестве делегата контроллера назначения.

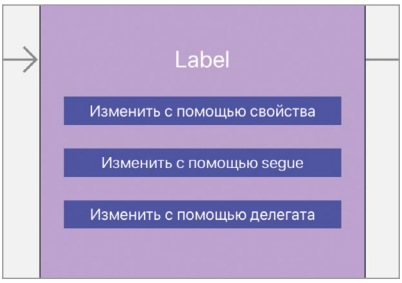

Рис. 10.13. Новая кнопка на сцене А

- В классе **ViewController** реализуйте экшн-метод **editDataWithDelegate** (листинг 10.15).

ЛИСТИНГ 10.15

```
// переход от А к Б
// передача данных с помощью свойства и установка делегата
@IBAction func editDataWithDelegate(_ sender: UIButton) {
    // получаем вью контроллер
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    let editScreen = storyboard.instantiateViewController(withIdentifier:
"SecondViewController") as! SecondViewController
```

```
    // передаем данные
    editScreen.updatingData = dataLabel.text ?? ""

    // устанавливаем текущий класс в качестве делегата
    editScreen.handleUpdatedDataDelegate = self

    // открываем следующий экран
    self.navigationController?.pushViewController(editScreen, animated:
true)
}
```

Тело метода **editDataWithDelegate** практически идентично методу **editDataWithProperty** за тем исключением, что помимо передачи данных в нем также устанавливается ссылка на класс-делегат.

- Свяжите метод **editDataWithDelegate** с кнопкой «**Изменить с помощью делегата**».

Теперь доработаем контроллер Б, реализовав вызов метода делегата при изменении данных.

- На сцене Б разместите новую кнопку «**Сохранить с помощью делегата**». Разместите ее ниже в ряд и оформите в соответствии со стилем остальных кнопок (рис. 10.14).

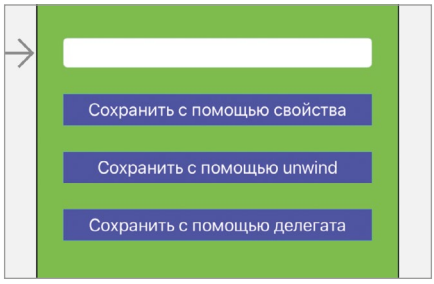

Рис. 10.14. Новая кнопка на сцене Б

- В классе **SecondViewController** реализуйте экшн-метод **saveDataWithDelegate** (листинг 10.16).

ЛИСТИНГ 10.16

```
// Переход от Б к А
// Передача данных с помощью делегата
@IBAction func saveDataWithDelegate (_ sender: UIButton) {
    // получаем обновленные данные
    let updatedData = dataTextField.text ?? ""
    // вызываем метод делегата
```

```
    handleUpdatedDataDelegate?.onDataUpdate(data: updatedData)
    // возвращаемся на предыдущий экран
    navigationController?.popViewController(animated: true)
}
```

- Свяжите нажатие кнопки «**Сохранить с помощью делегата**» с вызовом метода **saveDataWithDelegate**.
- Запустите приложение и попробуйте в действии реализованную функциональность.

Использование делегата – это очень эффективный способ передачи данных между контроллерами. Его самым важным плюсом является то, что контроллеры связываются на основе протокола и вся ответственность за обработку данных ложится на заинтересованный в этом контроллер (А). То есть на самом деле контроллеру Б вообще не важно, что будет с данными – он их обновил и далее передал контроллеру А, мол делай с ними все, что считаешь нужным.

Такой подход снижает связанность контроллеров и позволяет многократно повторно использовать их в различных сценариях. Так, например, мы с легкостью можем указать в качестве делегата другой контроллер, и это не вызовет каких-либо ошибок. Главное, чтобы он соответствовал протоколу.

Также вам стоит обратить внимание на то, что кнопка «**Сохранить с помощью делегата**» приводит к изменению данных только в том случае, если сцена Б была показана по нажатии кнопки «**Изменить с помощью делегата**». Это связано с тем, что в остальных случаях свойство **handleUpdatedDataDelegate** не будет содержать ссылку на делегат.

10.6 Передача данных от Б к А с помощью замыкания

Последним способом передачи данных между контроллерами, который мы реализуем в этой главе, является подход с использованием замыкания. Этот вариант очень похож на использование делегата, но вместо ссылки на делегат в контроллере Б будет храниться замыкание, вызываемое после изменения данных (рис. 10.15). Замыкание способно хранить ссылку на контроллер А, а значит и производить в нем работу по обработке обновленных данных, даже если сцена А в данный момент не видна на экране.

Значительный плюс такого подхода состоит в том, что замыкание может быть изменено в зависимости от контекста использования контроллера Б, в то время, как использование делегата подразумевает вызов одного и того же метода.

- В классе **SecondViewController** объявите свойство **completionHandler** (листинг 10.17).

Рис. 10.15. Схема обмена данными

ЛИСТИНГ 10.17

```
var completionHandler: ((String) -> Void)?
```

> **Примечание** Для хранения замыканий, вызываемых при завершении какой-либо задачи, очень часто используют параметры с именами **completionHandler**, **completionClosure**, просто **completion** или **handler**. Собственно, «completion handler» с английского переводится, как «обработчик завершения».

Тип замыкания **((String) -> Void)?** подразумевает, что если замыкание существует (значение свойства может быть **nil**), в него будет передано обновленное текстовое значение.

Теперь создадим на сцене А все необходимые для передачи замыкания элементы.

- На сцене А создайте кнопку с текстом «**Изменить с помощью замыкания**». Вновь, используя констрейнты, разместите ее в 30 точках ниже.
- Оформите кнопку в соответствии со стилем других кнопок.

При нажатии на кнопку «**Изменить с помощью замыкания**» (рис. 10.16) контроллер А будет передавать данные для изменения, а также инициализировать значение замыкания.

Рис. 10.16. Новая кнопка на сцене А

- В классе **ViewController** реализуйте экшн-метод **editDataWithClosure** (листинг 10.18).

ЛИСТИНГ 10.18

```
// переход от А к Б
// передача данных с помощью свойства и инициализация замыкания
@IBAction func editDataWithClosure(_ sender: UIButton) {
    // получаем вью контроллер
    let storyboard = UIStoryboard(name: "Main", bundle: nil)
    let editScreen = storyboard.instantiateViewController(withIdentifier:
"SecondViewController") as! SecondViewController

    // передаем данные
    editScreen.updatingData = dataLabel.text ?? ""

    // передаем необходимое замыкание
    editScreen.completionHandler = { [unowned  self] updatedValue in
        updatedData = updatedValue
        updateLabel(withText: updatedValue)
    }

    // открываем следующий экран
    self.navigationController?.pushViewController(editScreen, animated:
true)
}
```

Тело метода **editDataWithClosure** очень похоже на метод **editDataWithProperty** за исключением передачи замыкания.

- Свяжите метод **editDataWithClosure** с кнопкой «**Изменить с помощью замыкания**».

Теперь доработаем контроллер Б, реализовав вызов замыкания при изменении данных.

- На сцене Б разместите новую кнопку «**Сохранить с помощью замыкания**». Разместите ее ниже в ряд и оформите в соответствии со стилем остальных кнопок (рис. 10.17).
- В классе **SecondViewController** реализуйте экшн-метод **saveDataWithClosure** (листинг 10.19).

ЛИСТИНГ 10.19

```
// Переход от Б к А
// Передача данных с помощью замыкания
```

```
@IBAction func saveDataWithClosure(_ sender: UIButton) {
    // получаем обновленные данные
    let updatedData = dataTextField.text ?? ""
    // вызваем замыкание
    completionHandler?(updatedData)
    // возвращаемся на предыдущий экран
    navigationController?.popViewController(animated: true)
}
```

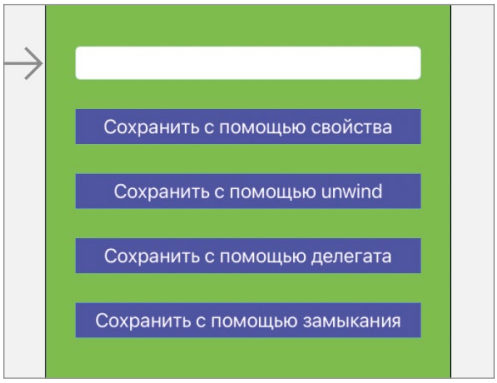

Рис. 10.17. Новая кнопка на сцене Б.

- Свяжите нажатие кнопки «**Сохранить с помощью замыкания**» с вызовом метода **saveDataWithClosure**.
- Запустите приложение и попробуйте в действии реализованную функциональность.

При использовании кнопки «**Сохранить с помощью замыкания**» данные успешно передаются в контроллер А только при условии, что переход к сцене Б был произведен с помощью кнопки «**Изменить с помощью замыкания**». В ином случае, в свойстве **completionHandler** находится **nil**.

10.7 Другие способы передачи данных

В этой главе мы рассмотрели несколько способов передачи данных внутри приложения, которые вы сможете использовать в своих будущих проектах. Но это далеко не все варианты, что может предложить вам Swift. Чем больший практический и теоретический опыт вы будете получать, тем больше новых способов работы с данными узнаете. В этом разделе я хотел бы упомянуть еще несколько важных моментов, связанных с передачей данных.

Использование ссылочных типов при передаче данных

В рассматриваемом материале для работы с данными мы используем тип данных **String**, который является значимым (value type). Но что, если вместо этого использовать ссылочный тип, например, такой:

```
class AppData {
    var data: String
    init(data: String) {
        self.data = data
    }
}
```

Данные из контроллера А в контроллер Б будут передаваться одним из рассмотренных способов, но обратной передачи не потребуется, так как оба контроллера будут ссылаться на одно и то же значение (рис. 10.18). Будет достаточно лишь обновить значение в свойстве контроллера Б, а при возвращении к сцене А обновить интерфейс с помощью метода **viewWillAppear**.

Рис. 10.18. Использование ссылочного типа данных

Работа с общим хранилищем

Одним из вариантов совместно работы с данными является использование различных хранилищ, например, User Defaults, когда каждый контроллер может самостоятельно подгружать из хранилища и обновлять требуемые ему данные. Нужно быть крайне осторожным, используя такой подход, так как в этом случае растет количество точек взаимодействия с хранилищем. Изменения данных в хранилище не всегда будут явными, а значит отслеживать их будет гораздо сложнее.

Использование паттерна «Одиночка»

Еще одним вариантом, который часто используют программисты, является применение шаблона проектирования «Одиночка» (Singleton). Напомню, что данный шаблон подразумевает единую точку входа в класс и один экземпляр этого класса на все приложение:

```
class User {
        static shared = User()
        var id: Int = 0
        init() {
                // ...
        }
}
```

Среди многих программистов данный шаблон считается антипаттерном. Его критикуют и не любят. Но лично я отношусь к нему более сдержанно и использую его в тех случаях, когда работа приложения очень тесно связана с экземпляром данного типа, и растрата памяти на его постоянное хранение оправдана.

Использование экземпляра класса AppDelegate

Иногда у вас может возникнуть желание хранить данные в классе **AppDelegate**, создав в нем несколько свойств и проинициализировав в них значения. Старайтесь избегать такого подхода. В задачи класса **AppDelegate** не входит обеспечение хранения — он предназначен для выполнения совершенно других задач.

Использование координаторов

Еще одним вариантом организации, о котором я хотел бы упомянуть, является использование координаторов. Я применял такой подход при разработке приложения «**Subs Tracker – Трекер подписок и регулярных платежей**». Это был учебный проект, созданный для изучения паттерна MVC, но с одним важным дополнением: в проекте используются так называемые координаторы.

Примечание Координаторы – это вовсе не мое изобретение. Они очень активно используются при разработке приложения командой Авито. На YouTube вы можете найти несколько прекрасных докладов по этой теме.

Координатор – это сущность, управляющая отображением сцен в приложении, а также обеспечивающая передачу и распространение данных внутри него. И таких координаторов в приложении может (и должно) быть несколько. Связь координаторов обычно представлена в виде дерева, где каждый координатор управляет собственным множеством сцен. Причем одни и те же сцены могут использоваться в различных координаторах (рис. 10.19).

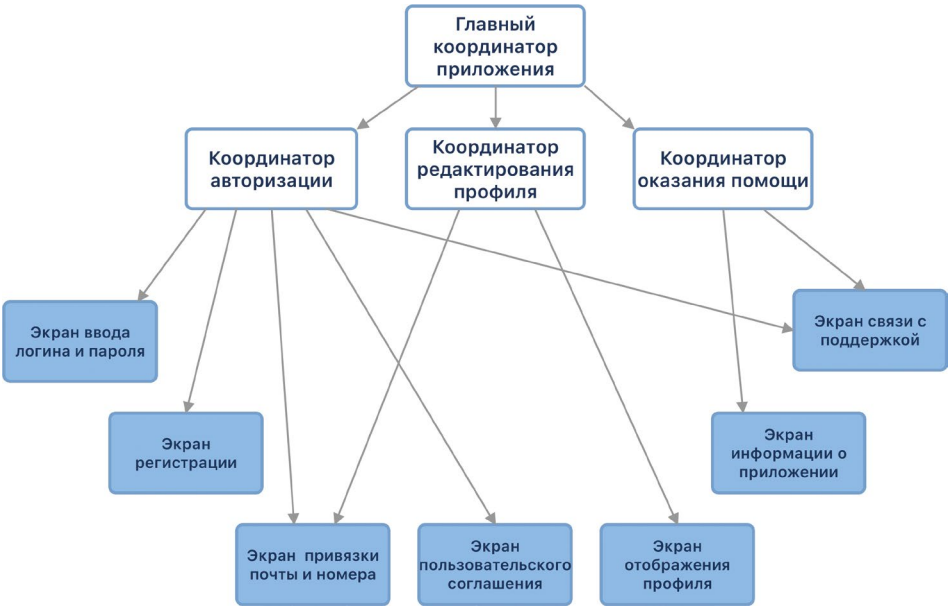

Рис. 10.19. Пример структуры координаторов и сцен

Для чего нужны координаторы?

Они сводят к минимуму связанность контроллеров, позволяя сделать их по-настоящему независимыми и переиспользуемыми. Каждый координатор предназначен для решения собственной задачи: для авторизации и регистрации, для управления профилем, для работы с товарными позициями и т.д. Примеров может быть бесконечное множество. Каждый координатор работает с большим количеством сцен. Но при этом они ничего не знают друг о друге – все взаимодействие между ними происходит через координаторы.

Если приложению требуется отобразить новую сцену, например, перейти от экрана авторизации к основному экрану со списком товаров, то решение этого

вопроса ложится на плечи координаторов, но не вью контроллеров. Если вам требуется распространить измененные данные (например, о добавленном в корзину товаре) в приложении, для этого также используются координаторы.

Координатор с программной точки зрения представляет из себя класс, подписанный на один или несколько специальных протоколов. Протоколы наделяют класс некой стандартной функциональностью, позволяющей работать с контроллерами и обмениваться данными.

> **Примечание** Конкретные реализации данных протоколов вы сможете найти в моем GitHub в исходном коде приложения Subs Tracker.

На этом мы завершаем изучение вопроса обмена данными между контроллерами внутри ваших приложений. В будущем вы неоднократно вернетесь к этой главе, так что советую оставить закладку.