Негосударственное образовательное учреждение
высшего профессионального образования
Московский технологический институт «ВТУ»

А.С. Бондаренко

Лабораторный практикум

по дисциплине

«Разработка мобильного приложения под платформу iOS»

Москва, 2014

Оглавление

Лабораторная работа № 1

«Установка среды разработки XCode, создание и настройка проекта»

Цель работы

1. Получить практические знания по установке IDE XCode.
2. Научиться создавать проект в IDE XCode.
3. Изучить настройки проекта и среды IDE XCode.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

Освоить лекционный материал по теме: «Изучение возможностей IDE XCode».

Теоретическая часть

Установка IDE XCode

XCode *– интегрированная среда разработки (IDE – integrated development environment), разработанная компанией Apple, содержит инструменты разработки программного обеспечения для операционных систем OS X и iOS.*

Прежде чем приступить к разработке приложений, необходимо настроить среду разработки для работы и убедиться, что имеются все необходимые инструменты.

Для разработки iOS-приложений понадобятся:

- компьютер с установленной операционной системой не ниже OS X 10.7 (Lion);
- XCode;
- iOS SDK.

Для установки XCode достаточно зайти в Mac App Store на компьютере (по умолчанию устанавливается вместе с OS X 10.7 и выше) под своей учетной записью, если нет учетной записи – завести ее. В поисковой строке указать ключевое слово «XCode», в поисковой выдаче найти данное приложение и нажать на кнопку «Бесплатно» («Free»). Начнется процесс скачивания программы в директорию */Applications*. После завершения скачивания необходимо запустить исполняемый файл.

На рисунке ниже представлен весь процесс установки XCode.

Рисунок 1. Процесс установки XCode

Создание проекта

Для с**оздания проекта в XCode можно воспользоваться двумя способами:**

1. При запуске XCode откроется диалоговое окно с предложением создать новый проект или открыть существующий из списка (см. рисунок ниже).

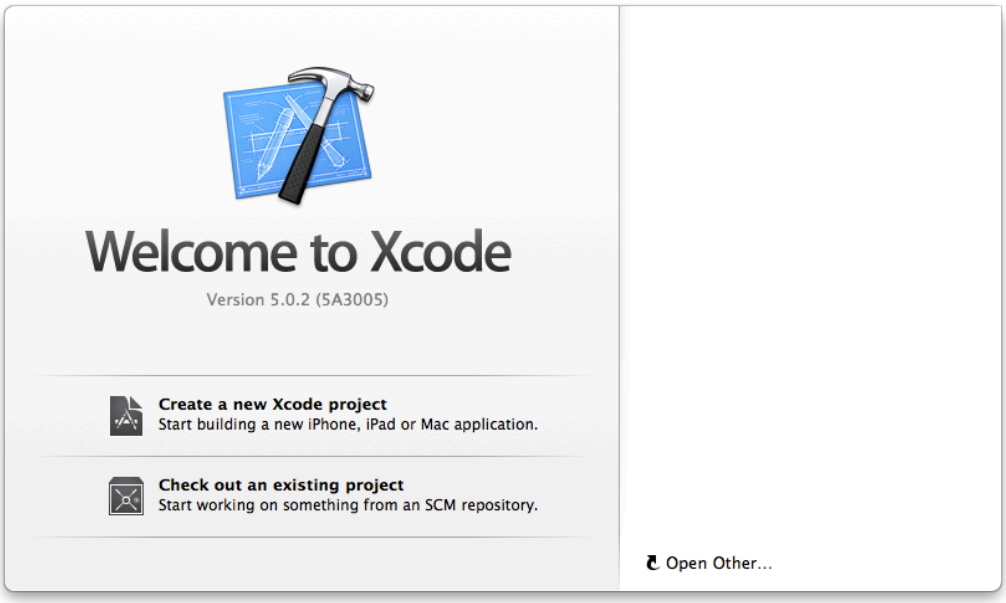


Рисунок 2. Диалоговое окно XCode «Создание проекта»

2. Через меню программы перейти по следующему пути: *File* → *New Project* (горячая клавиша Command-N) (см. рисунок ниже).

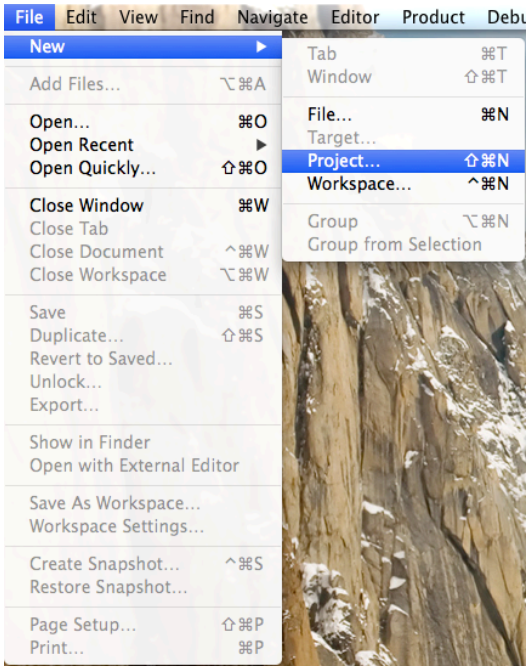


Рисунок 3. Создание проекта. Меню XCode

На следующем этапе необходимо выбрать необходимый шаблон проекта iOS-приложения (см. рисунок ниже).

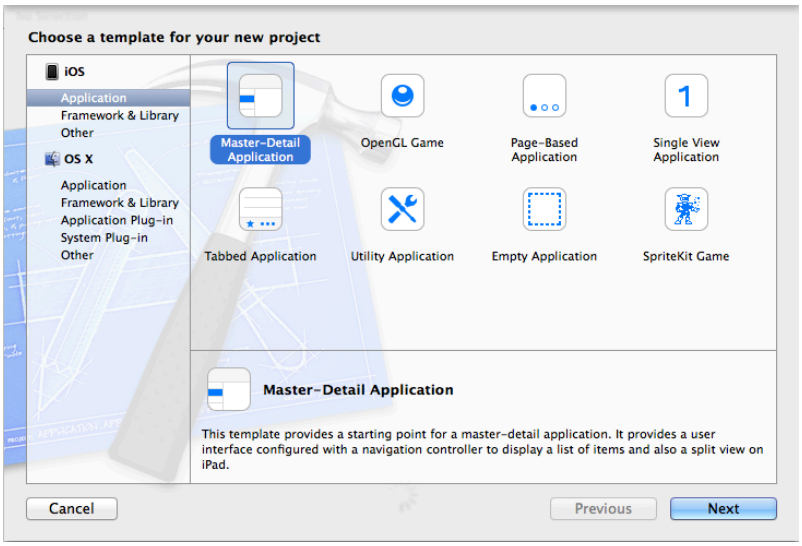


Рисунок 4. Окно выбора шаблона проекта

После выбора типа шаблона (например, Empty Application) необходимо перейти к форме заполнения данных о проекте (см. рисунок ниже).

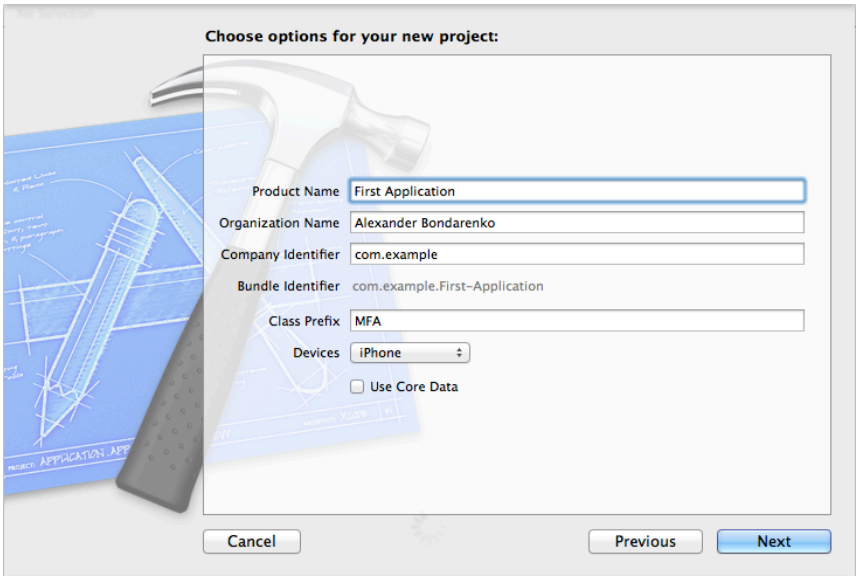


Рисунок 5. Заполнение данных о проекте

Информация, которую следует указать в форме заполнения данных о проекте, приведена на рисунке ниже.

Имя проекта	•XCode использует введенное имя для названия самого проекта и будущего приложения.
Идентификатор компании	•Уникальная строка, которую использует XCode, вместе с названием приложения, для создания идентификатора приложения. Указывается в случае, если идентификатор компании имеется. В противном случае можно использовать com.example.
Префикс для названия классов	•В Objective-C классы должны носить уникальные имена. •Для того чтобы использовать данные классы в других проектах (например, через фреймворки или статичные библиотеки), применяется следующая структура: «ПрефиксНазваниеТипКласса». Например: `MFAMainViewController`). •Стоит учитывать, что префиксы, состоящие из двух букв зарезервированы за компанией Apple (так она помечает свои классы в iOS SDK), поэтому необходимо использовать префиксы, состоящие из трех букв (например, MFA).
Выбор типа устройства	•При помощи выпадающего списка осуществляется выбор типа устройства, для которого разрабатывается приложение, – iPhone/iPad/Universal.
Использование Core Date	•Если указать, что в проекте будет использоваться Core Date, то в проекте автоматически создаются необходимые классы и методы для этого. •Классы и методы для использования Core Date можно добавить позднее самостоятельно.

Рисунок 6. Заполнение данных о проекте

После заполнения всех данных необходимо перейти к следующему этапу – выбору директории, где будет храниться проект (см. рисунок ниже). После нажатия кнопки **Create** проект считается созданным.

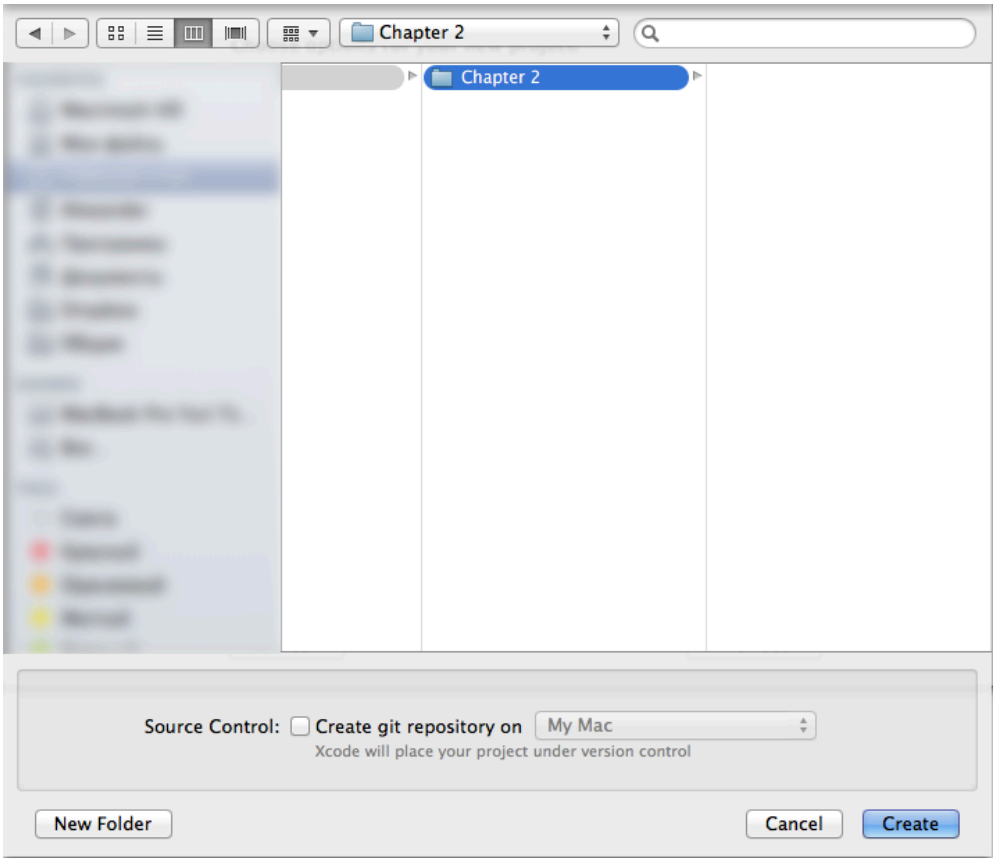


Рисунок 7. Выбор директории для хранения файлов проекта

Настройки проекта и среды

Для получения доступа к настройкам проекта необходимо в панели навигации в меню *Навигация по проекту* выбрать файл с именем проекта, после чего откроется редактор проекта (см. рисунок ниже).

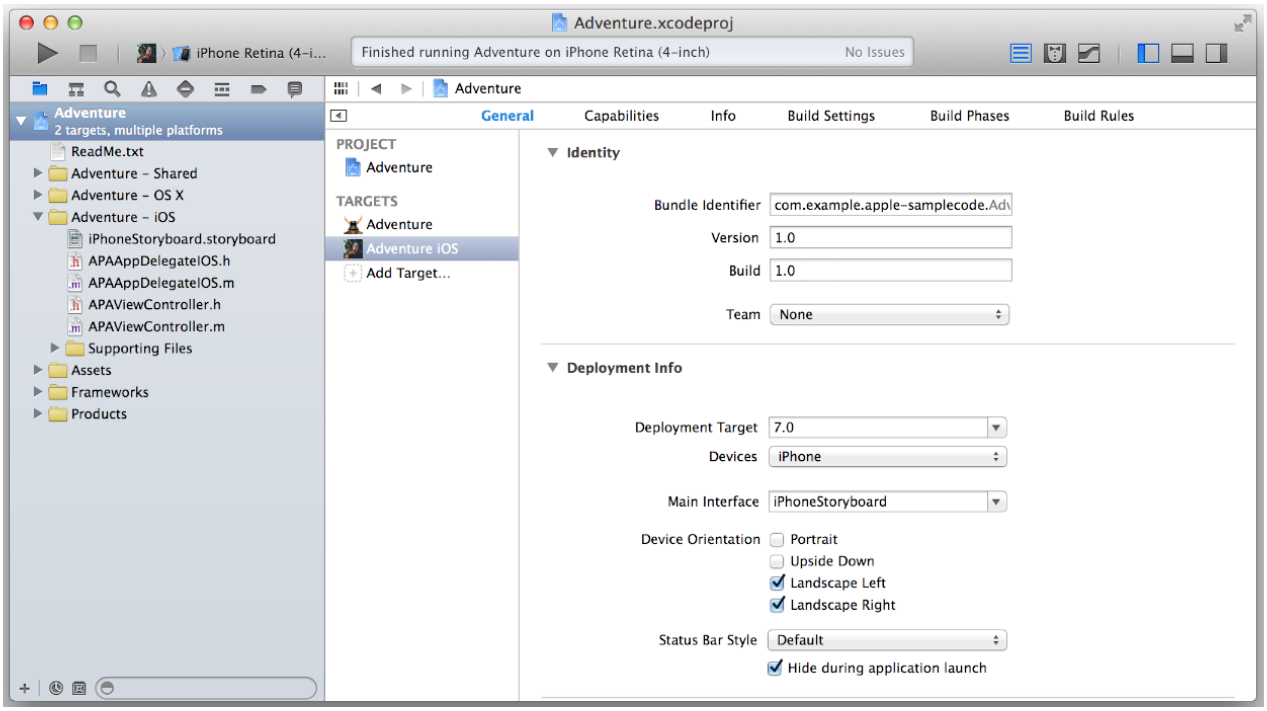


Рисунок 8. Настройки проекта

Все имеющиеся настройки можно разделить на две группы:

1. Настройки проекта (устанавливают правила для всего проекта и всех целей);
2. Настройка целей (уникальны в рамках конкретной цели).

Цели (targets) *– специальная конфигурация для создания и содержания инструкций при сборке проекта из определенного набора файлов или рабочего пространства.*

Настройки проекта состоят из **двух вкладок:**

1. **Вкладка Info** – на рисунке ниже приведены параметры, которые включает в себя данная вкладка.

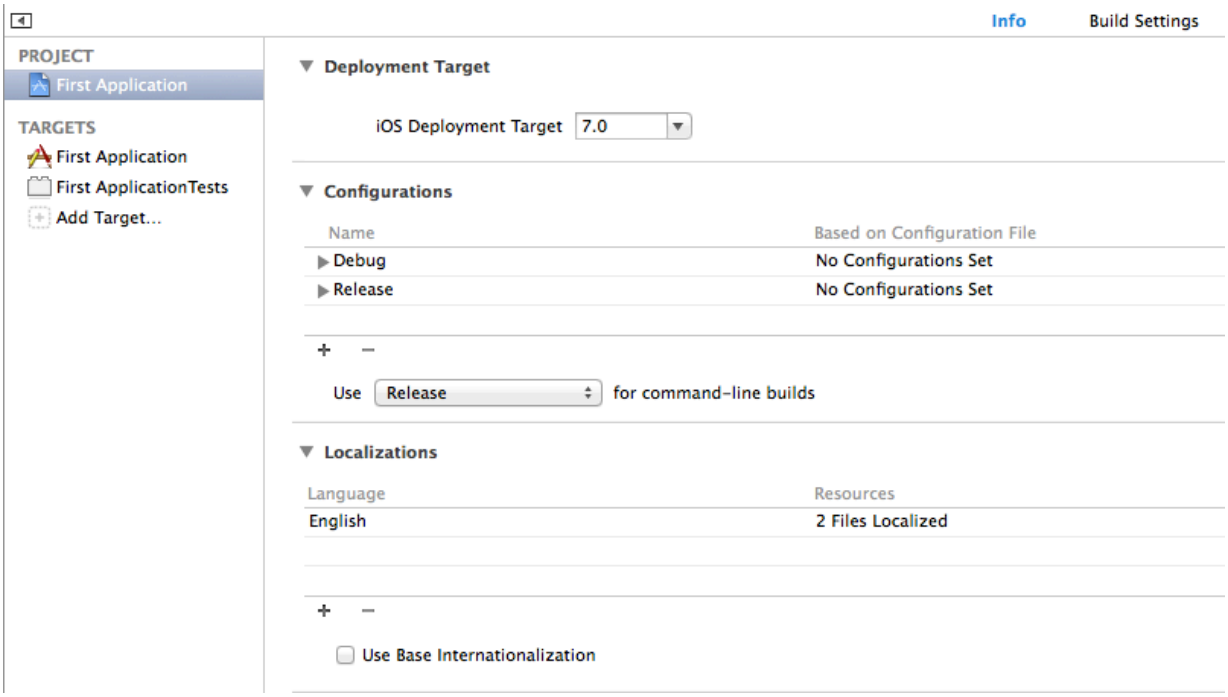


Рисунок 9. Настройки вкладки Info

В таблице ниже дано описание полей настроек вкладки Info.

Таблица 1.
Описание полей настроек вкладки Info

Название поля	Описание
Deployment Target	Выбор минимальной версии iOS для поддержки приложения.
Configurations	По умолчанию данный параметр имеет две конфигурации: Debug и Release. Конфигурация Debug используется для создания и отладки приложения (содержит дополнительную информацию для разработчика), конфигурация Release – для выпуска приложения (нет излишней информации).
Localizations	Установка локализации на необходимые языки (по умолчанию используется только английская).

2. **Вкладка Build Settings** – содержит большое количество полей для более тонкой настройки проекта (полную информацию можно получить на сайте) (см. рисунок ниже).

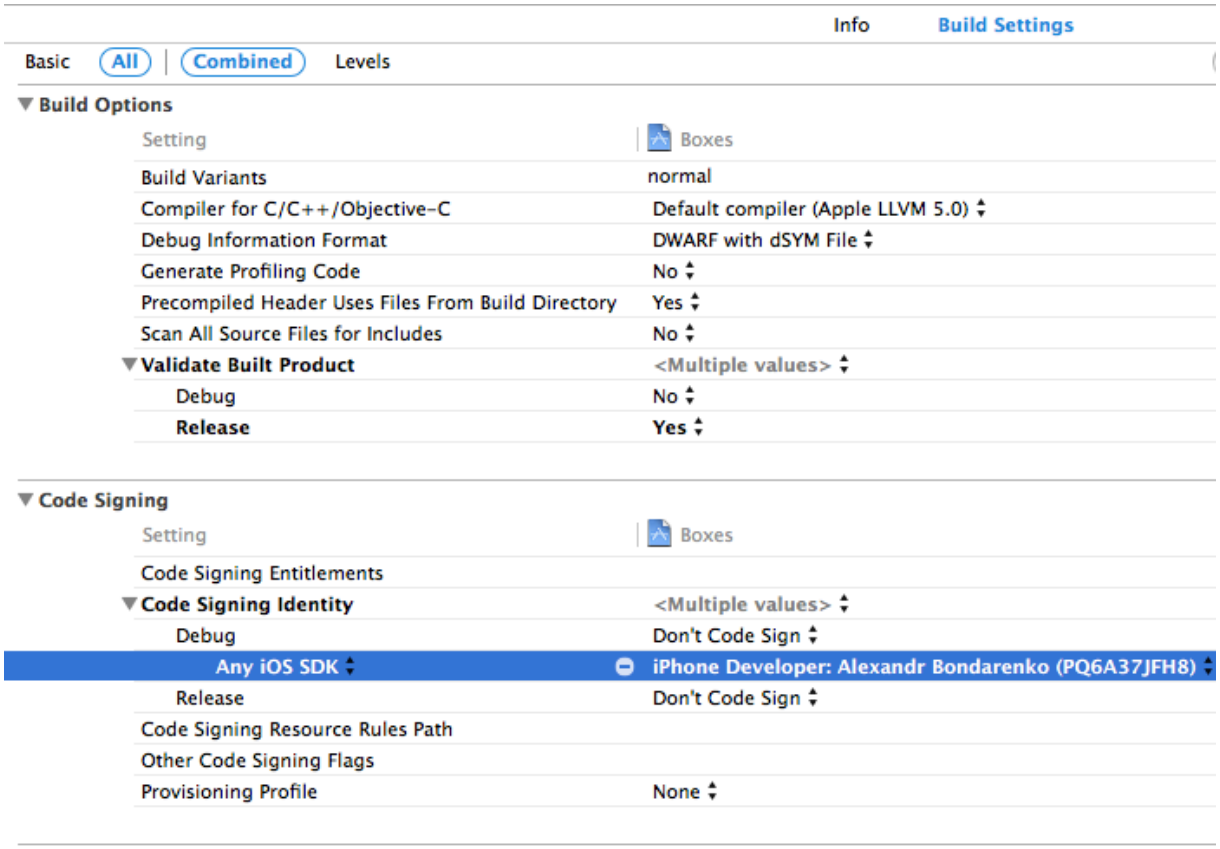


Рисунок 10. Настройки вкладки Build settings

В таблице ниже дано описание групп полей вкладки Build Settings.

Таблица 2.

Описание групп полей вкладки Build Settings

Название группы	Описание
Architectures	Позволяет выбирать базовую версию SDK для разработки, а также архитектуру устройства, для которого разрабатывается приложение.
Build Options	Позволяет настраивать пути в файловой системе для хранения файлов сборки.
Build Locations	Установка локализации на необходимые языки (по умолчанию используется только английская).
Code Signing	Позволяет настраивать в различных конфигурациях профили для установки приложения на физическое устройство.
Deployment	Позволяет настраивать дополнительные параметры сборки (например, выбор минимальной версии iOS).
Linking	Позволяет указать ссылки на сторонние модули или пути их раз-

	мещения в файловой системе.

Чтобы собрать и запустить iOS-приложение, необходимо выбрать схему и точку назначения, где будет запущено приложение. После чего нажать кнопку **Выполнить** на панели инструментов. Нажатие кнопки **Стоп** приведет к остановке выполнения и выходу из приложения (см. рисунок ниже).

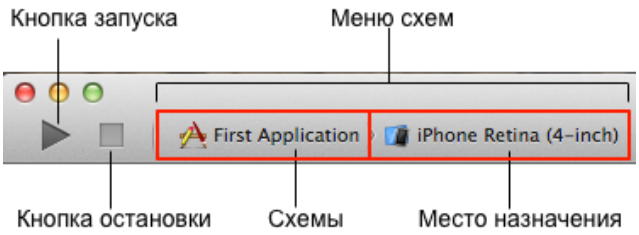


Рисунок 11. Меню схем на панели управления

Схемы *представляют собой набор параметров и настроек, которые применяются при сборке проекта.*

Диалоговое окно для управления схемами можно вызвать несколькими способами:

- командой главного меню **Product** → **Scheme** → **Manage Schemes**;
- командой **Manage Scheme** рядом с меню выбора схем (см. рисунок ниже).

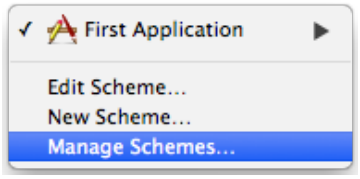


Рисунок 12. Выбор меню управления схемами

По умолчанию при открытии диалогового окна для управления схемами включена функция автоматической генерации схем (Autocreate schemas). Для ручного редактирования схем необходимо выбрать схему и нажать кнопку **Edit** (см. рисунок ниже).

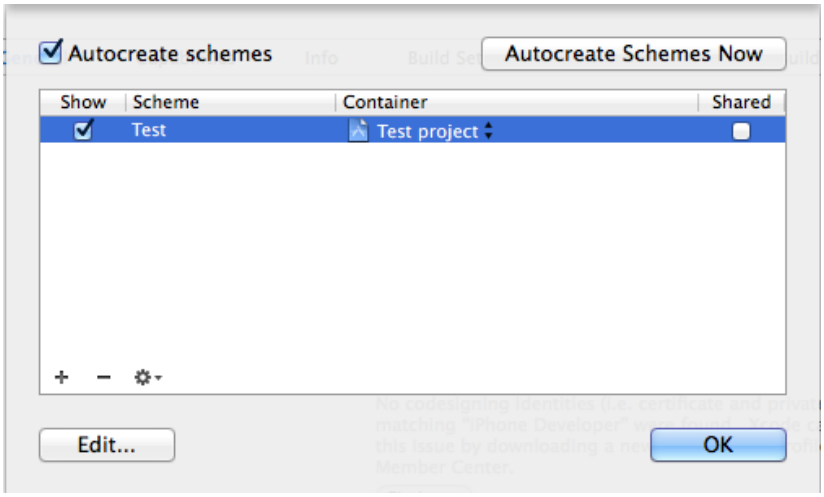


Рисунок 13. Диалоговое окно управления схемами

Редактировать схему целесообразно для выполнения следующих действий:

- сборка несколько целей (targets);
- выполнение сценариев (скриптов) до или после определенных действий;
- отправка email до или после определенных действий;
- запуск приложения с диагностикой управления памятью;
- выбор конфигурации для запуска между Debug и Release.

Для запуска приложения на устройстве в процессе разработки необходимо подключить устройство к компьютеру и добавить его в список тестирования данного приложения (provisioning profile).

Создать профиль, который позволит установить приложение на устройство, можно в XCode. Для этого необходимо перейти в меню *Window* → *Organizer*, открыть вкладку **Devices**. В левой части окна следует выбрать нужное устройство и нажать на кнопку **Use for Development**. Появится диалоговое окно, где следует отметить, в каких программах данное устройство нужно зарегистрировать, и нажать кнопку **Choose** (см. рисунок ниже).

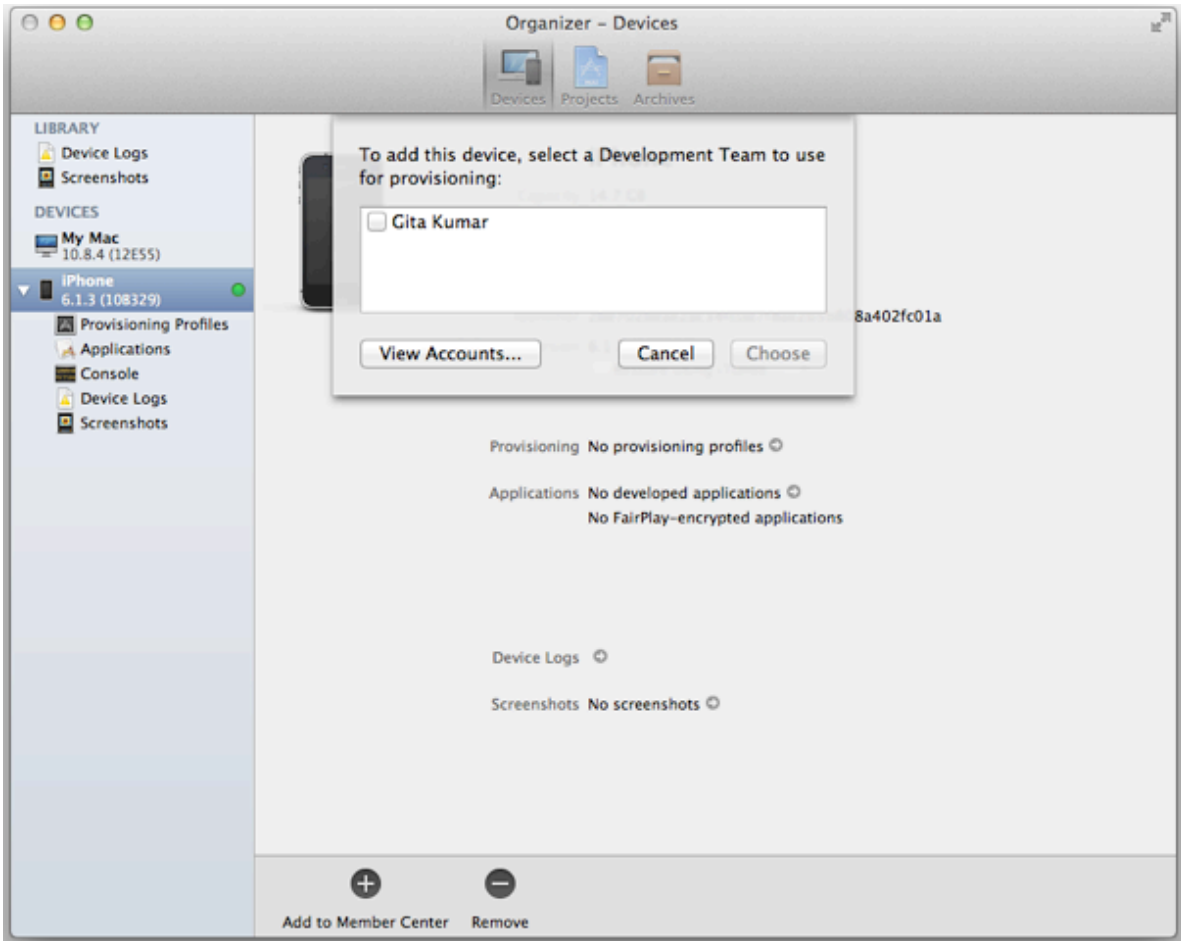


Рисунок 14. Регистрация устройства и формирование профиля

Практическая часть

Постановка задачи

Создать проект на основе шаблона. Настроить проект для различных конфигураций.

Порядок выполнения работы

1. Проанализировать задание, скачать, установить и запустить XCode.
2. Создать в XCode проект на основе шаблона iOS-приложения, категории Application.
3. Настроить проект согласно заданиям варианта.
4. Собрать проект и запустить на симуляторе.

Вариант 1

1. Создать универсальный проект на основе шаблона **Empty Application**, скомпилировать и запустить на симуляторе.
2. В настройках проекта установить:

- минимальную поддерживаемую версию iOS 6.0;

- создать на основе конфигурации Release, конфигурацию AdHoc;
- добавить поддержку локализации русского языка.

3. В настройках цели (targets) установить:

- версию приложения 0.1;
- версию сборки 1;
- минимальную поддерживаемую версию iOS 7.0;
- запретить ландшафтную ориентацию для iPhone;
- подключить фреймворк SystemConfiguration;
- задать имя приложения по следующей маске **Фамилия И.О.**

4. Отредактировать текущую схему, для действия запуска приложения, во вкладке **Diagnostics** включить флаг **Enable Zombie Objects**.
5. Создать новую схему с именем **AdHoc** и настроить таким образом, чтобы приложение собиралось в конфигурации **AdHoc**, при этом в данной схеме выключить флаг **Enable Zombie Objects**.

Вариант 2

1. Создать универсальный проект на основе шаблона **Tabbed Application**, скомпилировать и запустить на симуляторе.
2. В настройках проекта установить:

- минимальную поддерживаемую версию iOS 4.3;
- создать на основе конфигурации Release, конфигурацию AdHoc;
- добавить поддержку локализации украинского языка.

3. В настройках цели (targets) установить:

- версию приложения 0.2;
- версию сборки 2;
- минимальную поддерживаемую версию iOS 6.0;
- запретить портретную ориентацию для iPhone;
- подключить фреймворк CoreLocation;
- задать имя приложения по следующей маске **Фамилия И.О.**

4. Отредактировать текущую схему, для действия запуска приложения, во вкладке **Options** установить по умолчанию местоположение симулятора **Moscow, Russia**.
5. Создать новую схему с именем **AdHoc** и настроить таким образом, чтобы приложение собиралось в конфигурации **AdHoc**.

Вариант 3

1. Создать универсальный проект на основе шаблона **OpenGL Game**, скомпилировать и запустить на симуляторе.
2. В настройках проекта установить:
- минимальную поддерживаемую версию iOS 5.0;
- создать на основе конфигурации Release, конфигурацию AdHoc;
- добавить поддержку локализации испанского языка.
3. В настройках цели (targets) установить:
- версию приложения 0.3;
- версию сборки 3;
- минимальную поддерживаемую версию наследовать от настроек проекта;
- разрешить все возможные ориентацию для всех устройств;
- подключить фреймворк Social;
- задать имя приложения по следующей маске **Фамилия И.О.**
4. Отредактировать текущую схему, для действия запуска приложения, во вкладке **Options** запретить для симулятора определение местоположения.
5. Создать новую схему с именем **AdHoc** и настроить таким образом, чтобы приложение собиралось в конфигурации **AdHoc**.

Вариант 4

1. Создать универсальный проект на основе шаблона **SpriteKit Game**, скомпилировать и запустить на симуляторе.
2. В настройках проекта установить:
- минимальную поддерживаемую версию iOS 5.1;
- создать на основе конфигурации Release, конфигурацию AdHoc;
- добавить поддержку локализации японского языка.
3. В настройках цели (targets) установить:
- версию приложения 0.4;
- версию сборки 4;
- минимальную поддерживаемую версию iOS 6.0;
- запретить портретную ориентацию для iPad;
- подключить фреймворк OpenAL;
- задать имя приложения по следующей маске **Фамилия И.О.**
4. Отредактировать текущую схему, для действия запуска приложения, во вкладке **Diagnostics** включить флаг **Garbage Collection Activity**.

5. Создать новую схему с именем **AdHoc** и настроить таким образом, чтобы приложение собиралось в конфигурации **AdHoc**, при этом в данной схеме выключить флаг **Garbage Collection Activity**.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами представить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Для каких целей предназначен каждый шаблон iOS-приложения?
2. Что такое схемы в XCode и для чего они нужны?
3. Какие требования необходимо выполнить, чтобы установить приложение на устройства?

Литература

1. Дэйв Марк, Джек Наттинг, Джефф Ламарш, Фредерик Олссон. iOS 6 SDK. Разработка приложений для iPhone, iPad и iPod touch - Вильямс, 2013. – 672 с.
2. Neil Smyth. iOS 7 App Development Essentials: Developing iOS 7 Apps for the iPhone and iPad – eBookFrenzy, 2013. -736 с.
3. Ресурс разработчиков Apple.
4. Документация Apple по IDE XCode.
5. Документация Apple по Information Property List.

Лабораторная работа № 2
«Знакомство с Objective-C»

Цель работы

1. Изучить синтаксис Objective-C.
2. Изучить методы в Objective-C.
3. Изучить типы данных и свойства переменных.
4. Изучить классы в Objective-C

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

Освоить лекционный материал по теме: «Изучение возможностей IDE XCode».

Теоретическая часть

В языке Objective-C существует несколько основных типов данных: **int, float, double, char, id, BOOL, unsigned, void, указатели, структуры**. В Obj-C любое число, символ или строка символов называется константой (constant).

Для вывода информации в консоль среды разработки используется функция `NSLog();`.

В таблице ниже представлены наиболее часто используемые спецификаторы для вывода объектов или значений

Таблица 3.
Спецификаторы функции NSLog()

Спецификатор	Пояснения
`NSLog (@"Object: %@", anObject);`	`%@` служит для вывода объектов.
`NSLog (@"Integer: %i", aInt);` `NSLog (@"Integer: %d", aInt);`	`%i` или `%d` служат для вывода целочисленных значений. Вариации: `%20d` – резервирование 20 количества символов.

NSLog (@"Object: %f", aFloat);	`%f` служит для вывода вещественных значений. Вариации: `%.2f` – вывод значения с двумя знаками после запятой.

Полная информация об используемых спецификаторов представлена в документации Apple.

*Текст между символами @" " называется **строкой**.*

Строка может иметь длину ноль и более символов. Функция `NSLog` имеет поддержку следующих возможностей:

- корректная обработка управляющих символов (знак «\» – обратный слеш – служит сигналом о том, что следующий символ за ним не обычная буква, а специальный символ);
- отображение нескольких значений или сочетаний значений, например:

```
NSLog(@"Integer = %d, \n Float = %.2f, \n String = %@, 12, 24,543, @"String").
```

Чтобы заставить объект выполнить какое--либо действие, нужно отправить ему сообщение. Отправка сообщения выглядит следующим образом:

```
[receiver message];
```

В сообщении можно передавать параметры для вызова метода:

```
[receiver message: param1];
```

Если параметров больше одного, то они отделяются двоеточием. Количество двоеточий равно количеству параметров, например:

```
[receiver message: param1 :param2];
```

На практике чаще используется поименование параметров, что значительно упрощает чтение кода:

```
[receiver message: param1 withAnyObject: param2]; //сообщение в канонической форме.
```

Отправка сообщения, как и любая функция в ООП, возвращает определенное значение:

```
NSInteger intValue;
intValue = [receiver message];    //age];ноет:  param2];  Object:  param2];
целочисленное значение.
```

При отправке сообщения объекту, который принадлежит классу, нереализовавшему заказанный метод, возникает исключение. Для проверки – отвечает ли данный объект на какое-либо сообщение – можно использовать следующий шаблон кода:

```
if([anObject respondsToSelector: @selector(myMethod: )])
{
// ])hod: )])yMethod: )])ctor(myMeth
[anObject myMethod: param1];
}
else
{
//semhod: )])yMethod: )])ctor(myMeth
}
```

Метод может возвращать тип данных `void`, в этом случае вызывающее его сообщение ничего не возвращает. Если явно не указывать тип возвращаемого значения, сообщение возвращает значение типа `id`.

*Действия, которые могут выполнять объекты в языке Objective-C, называются **методами**.*

Синтаксис метода выглядит следующим образом:

```
-(<тип данных возвращаемого значения>) основнаяЧастьИмениМетода:
(<тип данных первого параметра>)имяПервогоПараметра
дополнительнаяЧастьИмениМетода:
(<тип данных второго параметра>)имяВторогоПараметра
```

Сигнатура метода будет выглядеть следующим образом:

```
 "-основнаяЧастьИмениМетода:дополнительнаяЧастьИмениМетода:"
```

Пример объявления метода:

```
-(void)myMethodWitnParam:(id)Param1 andOneMore:(id)Param2;
```

Метод начинается либо со знака "–", который сообщает компилятору, что данный метод является **методом экземпляра данного класса,** так как может быть вызван только для объекта какого-нибудь класса. Либо со знака "+", данные методы являются **методами класса**.

После знака минуса или плюса в скобках указывается тип данных возвращаемого значения.

Затем следует имя метода (этика разработки на Objective-C требует давать исчерпывающие информативные имена, начинать с маленькой буквы и заканчивать названием принимающего параметра или намеком на него).

Если метод должен принимать параметры, то после имени метода ставится двоеточие.

Затем в скобках указывается тип данных параметра.

И в конце имя переменной параметра, первый параметр называется «**прямой параметр**».

Если требуется передать больше параметров, то через пробел указывается дополнительная часть имени метода (не обязательно, но желательно), двоеточие, тип данных и имя переменной.

Каждый метод содержит два невидимых параметра:

1. **Параметр `self`** – является аналогом *this* в C++, т.е. указывает на сам объект – получатель сообщения.

Данный аргумент (см. пример ниже) может использоваться для отправки сообщения самому себе (т.е. при обращении к собственным методам класса). Кроме параметра `self` используется еще одно зарезервированное слово – `super` – указатель на экземпляр базового класса. Вызывая `super`, вызываются родительские методы, не переопределенные данным классом.

```
[self myMethod];
```

2. **Параметр `_cmd`** – содержит селектор метода из глобальной таблицы селекторов.

> *Под **селектором** понимается имя метода или сообщения Objective-C, которые используются в исходном коде (в терминах языка программирования Objective-C).*

Всем методам, имеющим одинаковые имена, компилятор назначает один и тот же селектор.

По селектору метода можно получить адрес реализующей его функции. Такая функция отличается от описания метода только вставкой в начало списка аргументов двух дополни-

тельных параметров: указателя на сам объект (`self`) и селектора данного метода (`_cmd`). Послав объекту сообщение `methodForSelector:`, в ответ придет адрес, реализующий данный метод функции. Это позволит при необходимости многократного вызов одного и того же метода у заданного объекта сократить расходы ресурсов, возникающих при пересылки сообщений.

Если в начале метода поставить знак “+”, то такой метод будет являться **методом класса** и посылается классу (аналогичен объявлению `static` функции в C++). Соответственно, метод класса уже не будет применять неявный параметр `self`. Зачастую метод класса используется для инициализации объекта (см. пример ниже).

```
//прототип метода
+(id)alloc;
//отправка сообщения данному методу
[Class alloc];
```

Для создания нового класса в проекте можно воспользоваться командой *File → New → File* (горячая клавиша ⌘+N). Появится окно, в котором следует выбрать Objective-C class (см. рисунок ниже).

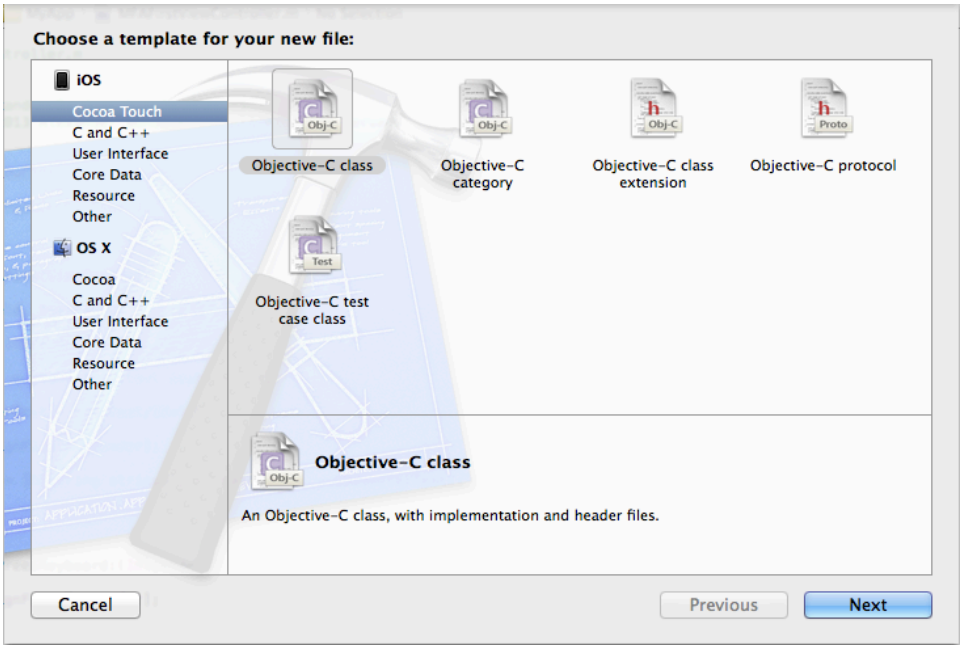


Рисунок 15. Создание нового класса в Objective-C

Объявление класса размещается в заголовочном файле между ключевыми словами `@interface` и `@end`:

```
//в файле “MAClass.h”
```

```
@interface MAClass:NSObject
{
      //объявление данных (инварианты)
}

//объявление методов и свойств

@end
```

Вместо конструкторов и деструкторов в Objective-C используются инициализаторы и деаллокаторы. Это обычные методы. Два из таких методов, определенных в `NSObject`, приведены ниже.

```
-(id)init
{
      if (self=[super init])
      {
            // инициализация переменных и свойств
      }
      return self;
}

-(void)dealloc
{
      [super dealloc];
}
```

В классе предке они выполняют функцию выделения памяти объекту и возвращения ее программе. Ниже представлены методы, которые должны в обязательном порядке вызываться в инициализаторах и деалокаторах, соответственно.

```
self = [super init] // в инициализаторе
[super dealloc] // в деалокаторе
```

В связи с тем, что все объекты распределяются в динамической памяти, создание объекта происходит в два этапа (см. рисунок ниже).

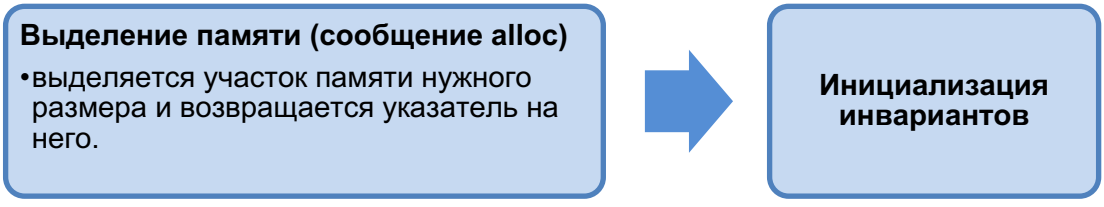


Рисунок 16. Этапы создания объекта

```
MAClass *anyObject = [[MACLass alloc] init];
```

Существует более удобный способ выделения памяти – использование метода `new`, который сочетает в себе методы `alloc` и `init`.

```
MAClass *anyObject = [MAClass new];
```

После создания объекта можно начинать работу с ним.

Практическая часть

Постановка задачи

Создать класс Objective-C, выполняющий указанное действие. Результаты вывести в консоль.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить класс Objective-C.
2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Необходимую информацию вывести в консоль при помощи функции `NSLog()`.
5. Собрать проект и запустить на симуляторе.

Вариант 1

1. Создать класс и задать ему имя по следующей маске – `FIOTime` (где FIO – инициалы автора).
2. Класс должен содержать три поля типа `NSInteger`, предназначенные для хранения часов, минут и секунд.
3. Инициализация данных должна происходить следующими способами:

- нулевыми значениями;
- заданным набором значений;
- текущим системным временем (см. подсказку ниже);

```
    NSDate *curentDate = [NSDate date];
    NSCalendar *gregorianCalendar = [[NSCalendar alloc] initWithCalen-
darIdentifier:NSGregorianCalendar];
    NSDateComponents *dateComponents = [gregorianCalendar compo-
nents:NSHourCalendarUnit | NSMinuteCalendarUnit | NSSecondCalendarUnit
fromDate:curentDate];
```

```
    NSLog(@"%i:%.2i:%i", [dateComponents hour], [dateComponents minute],
[dateComponents second]);
```

4. Создать следующие методы:

- метод, который будет выводить в консоль значения полей в формате hh:mm:ss;
- метод, который будет выводить в консоль значения полей в формате hh:mm:ss AM/PM;
- метод, который будет возвращать сумму значений двух объектов типа `FIOTime`, передаваемых в качестве аргументов;
- метод, который будет возвращать разность значений двух объектов типа `FIOTime`, передаваемых в качестве аргументов.

5. В функции `main()` следует создать два инициализированных объекта (один с текущим системным временем, другой с заданным набором значений) и один неинициализированный объект. Затем сложить два инициализированных значения и результат присвоить третьему объекту. Вывести в консоль значение третьего объекта в формате hh:mm:ss и hh:mm:ss AM/PM. Затем присвоить третьему объекту результат разности двух инициализированных объектов и так же вывести третий объект в консоль.
6. Привести методы класс `FIOTime` в соответствие правилами вычисления времени:

- при сложении секунд, минут их суммарное значение не должно быть более 60, при этом повышать значение следующего разряда.

Например: *если количество секунд получилось равным 60, то записывать значение 0, а минутам добавить единицу.*

- при вычитании секунд, минут их разностное значение не должно быть отрицательным, при этом понижать значение следующего разряда.

Например: *если количество секунд получилось равным 10, то записывать значение 50, а у минут отнять единицу.*

Вариант 2

1. Создать класс и задать ему имя по следующей маске – `FIOCoordinate` (где FIO – инициалы автора).
2. Класс должен содержать три поля (пример данных - 43°46,7’ S, т.е. 43 градуса 46,7 минут южной широты):

- тип `NSInteger` предназначенный для хранения числа градусов;
- типа `CGFloat`, предназначенный для хранения числа;
- тип `char`, предназначенный для указания направления (широты или долготы).

3. Инициализация данных должна происходить следующими способами:

- нулевыми значениями;
- заданным набором значений;

4. Создать следующие методы:

- метод, который будет выводить в консоль значение координат в градусах и минутах с десятичной дробью (пример: 57°23,4’ N);
- метод, который будет выводить в консоль значения координат в градусах в виде десятичной дроби (пример: 43,567891°);
- метод, который будет выводить в консоль значения координат в градусах, минутах и секундах с десятичной дробью (пример: 21°59’20,99” E);
- метод, который будет возвращать расстояние между двумя координатами на плоскости.

5. В функции `main()` следует создать координаты двух точек с любыми значениями. Вывести в консоль значение координат данных точек тремя способами. Затем рассчитать расстояние по прямой между данными точками, результат так же вывести в консоль.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. В чем разница между методами класса и методам экземпляра класса?
2. Что такое протоколы в Objective-C?
3. Основные типы данных в Objective-C?

Литература

1. Стефан Кочан. Программирование на Objective-C 2.0. Издательство: ЭКОМ Паблишерз, 2010.
2. Ресурс разработчиков Apple.
3. Документация Apple по Objective-C.

Лабораторная работа № 3

«Классы для хранения данных в Objective-C»

Цель работы

1. Изучить класс для работы со строками.
2. Изучить класс для работы с множеством.
3. Изучить класс для работы с массивами.
4. Изучить класс для работы со словарями.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

Освоить лекционный материал по теме: «Язык программирования Objective-C».

Теоретическая часть

Классы, представляющие данные (динамические массивы, наборы, словари, численные значения и т.п.), являются самыми используемыми в фреймворке Foundation. К таким классам относятся, например, `NSData`, `NSArray`, `NSString`, `NSDate`, `NSSet` и мн. др.

Классы, представляющие данные, делятся на **две категории:**

1. **`Mutable` – изменяемые классы.**

Объекты такого типа могут менять содержащееся в них значение. Имена изменяемых классов содержат слово **`Mutable`** (после префикса NS и перед именем неизменяемого класса, который они «расширяют»), например:

```
NSMutableData, NSMutableString, NSMutableArray.
```

2. `Immutable` – неизменяемые классы.

Изменяемые объекты в библиотеке `Foundation` наследуются от неизменяемых классов.

Неизменяемые классы не содержат никакого специального слова в своих названиях:

```
NSData, NSString, NSArray.
```

Имеются классы, задача которых состоит исключительно в хранении значений (класс `NSValue` и его подклассы, например, `NSNumber`), у которых нет изменяемых вариантов.

***Класс `NSString`** – класс, в экземплярах которого хранится последовательность символов.*

Для создания экземпляра класса `NSString` можно использовать запись вида:

```
NSString *aString = @”Hello”;
```

Или создать класс `NSString` при помощи метода класса `stringWithFormat:`

```
NSString *aString = [NSString stringWithFomat:@”Hello, %@”, Alex];
```

В целом, большая часть методов класса `NSString` отвечает за создание строк, и в разных ситуациях следует использовать необходимый метод.

Базовые методы (`length` и `characterAtIndex:`) возвращают количество символов (Unicode) и сами символы (Unicode).

***Классы `NSArray` и `NSMutableArray`** используются для хранения упорядоченных коллекций любых объектов-наследников `NSObject`, не обязательно однородных.*

Создать `NSArray` можно несколькими способами, которые делятся на три большие группы (см. рисунок ниже).

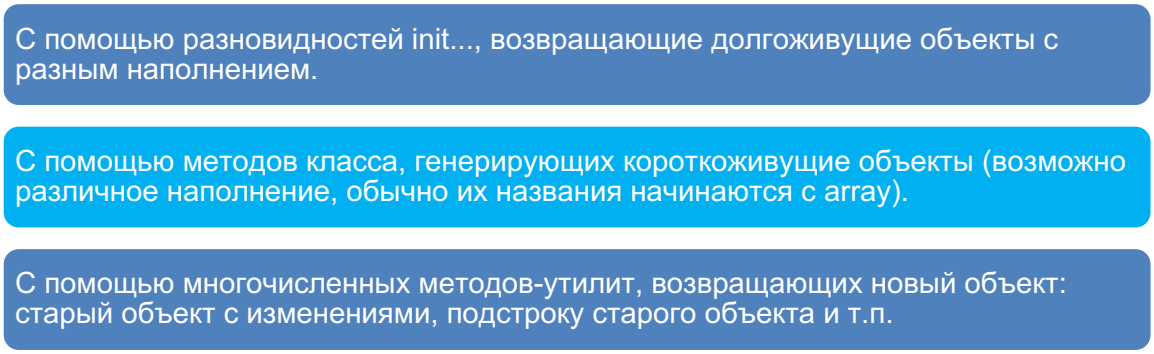


Рисунок 17. Способы создания NSArray

`NSMutableArray` наследует все методы класса `NSArray` и добавляет к ним методы для добавления и удаления объектов.

Классы `NSSet`, `NSMutableSet` и `NSCountedSet` *– неупорядоченные коллекции. Их элементы должны быть объектами класса-наследника `NSObject`, а также в этих объектах должны быть реализованы методы `isEqual:` и `hash`.*

Для размещения изменяемых объектов в множество их необходимо предварительно преобразовать в неизменяемые объекты.

При создании множества и при изменениях изменяемых множеств любое число равных (`isEqual:`) объектов входит в его состав только один раз. В этом заключается **главная особенность множеств.**

`NSCountedSet`, подкласс `NSMutableSet` запоминает, сколько раз в него были добавлены взаимно равные объекты.

Классы NSDictionary и NSMutableDictionary *– неупорядоченные коллекции, в которых каждому элементу (значению) соответствует уникальный ключ.*

Элементом словаря может быть объект любого подкласса `NSObject`; ключом – объект любого подкласса `NSObject`, подписавший контракт с протоколом `NSCopying`. В этом протоколе содержится только один обязательный метод – `copyWithZone`. Большие объекты не рекомендуется использовать в качестве ключей.

Основные методы неизменяемых словарей – `count`, `objectForKey:` и `keyEnumerator`. Присвоение нового значения изменяемому словарю производится с использованием метода `setObjectForKey:`, удаление значения – методом `removeObjectForKey:`.

С помощью методов `allKeys`, `allValues` и `allKeysForObject:` из словаря в виде `NSArray` можно получить полный список ключей, значений или ключей, соответствующих некоторому значению.

Практическая часть

Постановка задачи

Создать класс Objective-C, выполняющий указанное действие. Результаты вывести в консоль.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить класс Objective-C.

2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Необходимую информацию вывести в консоль при помощи функции `NSLog()`.
5. Собрать проект и запустить на симуляторе.

Вариант 1

1. Создать множество и заполнить данными при помощи генератора случайных чисел из 40 элементов (не повторяющимися целыми числами в диапазоне от 1 до 60). Вывести в консоль:

- количество элементов в множестве;
- элементы множества;
- сумму элементов множества;
- среднее арифметическое элементом множества.

2. Создать словарь и заполнить его данными из предыдущего множества следующим образом:

- под элементом словаря с ключом «three» должен находиться массив, который содержит числа, делящиеся без остатка на 3;
- под элементом словаря с ключом «order» должен находиться массив, содержащий числа расположенные в порядке возрастания;
- под элементом словаря с ключом «even» должен находиться массив, содержащий четные числа;
- под элементом словаря с ключом «odd» должен находиться массив, содержащий нечетные числа;
- под элементом словаря с ключом «four» должен находиться массив, в котором все числа, оканчивающиеся на цифру 4, уменьшены вдвое;
- под элементом словаря с ключом «range» должен находиться массив, в котором переставлены числа в порядке убывания.

3. Вывести элементы словаря отдельно по каждому ключу в консоль.

Вариант 2

1. Создать массив и заполнить данными из 40 элементов случайным образом, используя следующие требования к данным массива:

- вещественными значениями, лежащими в диапазоне от 0 до 1;
- вещественными знамениями x ($5 \le x < 6$);

- вещественными значениями x (-10 ≤ x <10);
- целыми значениями x (-60 ≤ x < 60);
- натуральными числами, делящимися нацело на 11 и 19 и находящимися в интервале от 20 до 150.

2. Вывести в консоль:
- количество элементов в массиве;
- элементы массива.

3. Создать словарь и заполнить его данными из предыдущего массива следующим образом:
- под элементом словаря с ключом «sum» должна находиться сумму всех элементов;
- под элементом словаря с ключом «multi» должно находиться произведение всех элементов;
- под элементом словаря с ключом «square» должна находиться сумму квадратов всех элементов;
- под элементом словаря с ключом «average» должно находиться среднее арифметическое всех элементов;
- под элементом словаря с ключом «max» должен находиться максимальный элемент;
- под элементом словаря с ключом «min» должен находиться минимальный элемент;
- под элементом словаря с ключом «unique» должен находиться массив, в котором удалены повторяющиеся элементы, а оставшиеся элементы выставлены в порядке возрастания;

4. Вывести элементы словаря отдельно по каждому ключу в консоль.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Какие классы для хранения данных присутствуют в Objective-C? В чем их назначение?
2. Что такое неизменяемые классы? В чем их особенность? Что такое изменяемые классы?

Литература

1. Стефан Кочан. Программирование на Objective-C 2.0. Издательство: ЭКОМ Паблишерз, 2010.
2. Ресурс разработчиков Apple.
3. Документация Apple по Objective-C.

Лабораторная работа № 4

«Библиотека UIKit, классы для создания пользовательского интерфейса»

Цель работы

1. Получить практические знания работы с библиотекой UIKit.
2. Научиться работать с классом `UIView` и с группой классов, наследуемых от `UIView`.
3. Научиться работать с классом `UIViewController` и с группой классов, наследуемых от `UIViewController`.
4. Получить практические знания работы с табличными представлениями.
5. Научиться создавать интерфейс приложения с использованием визуального редактора проектирования.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Фреймворк Cocoa Touch, изучение UIKit, концепция MVC».

Теоретическая часть

Изначально фреймворк Cocoa Touch создавался с применением парадигмы MVC, представляющей собой очень логичный способ разделения кода, лежащего в основе приложений с графическим пользовательским интерфейсом.

Шаблон MVC разделяется по функциональным возможностям на три разные категории (см. рисунок ниже).

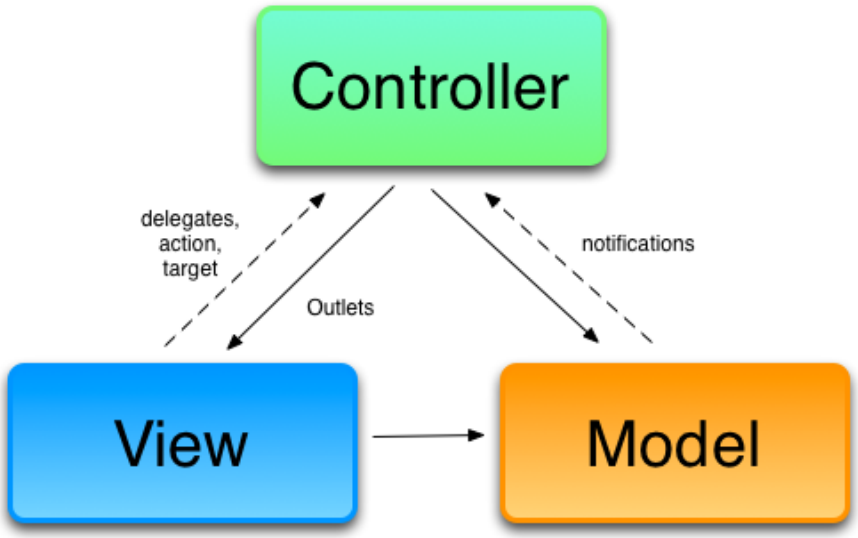


Рисунок 18. MVC в Cocoa Touch

- **Модель**. Состоит из классов, в которых хранятся данные приложения.
- **Представление**. Создает окна, элементы управления и другие элементы, которые пользователь видит и с которыми взаимодействует.
- **Контроллер**. Связывает модель и представление, реализует логику приложения, в соответствии с которой приложение обрабатывает данные, введенные пользователем.

Цель концепции MVC *– создать максимально независимые друг от друга объекты, реализующие все категории шаблона MVC.*

Любой объект, создаваемый разработчиком, должен четко идентифицироваться как объект, принадлежащий одной из перечисленных выше категорий. При этом он не должен иметь вообще или иметь минимальное количество функциональных возможностей, которые можно было бы отнести к остальным двум категориям.

Класс `UIView` является базовым классом UI-объектов, с которыми взаимодействует пользователь, он отвечает за визуализацию данных. UIKit представляет обширный набор подклассов `UIView` для отображения данных (см. таблицу ниже).

Таблица 4.

Наследуемые классы `UIView` библиотеки UIKit

Название класса	Описание
`UIWindow`	Определяет объект `window`, который управляет и координирует объекты класса `view` приложения, отображает данные объекты на экране.

UILabel	Реализует представление текстовой надписи, предназначенной только для чтения. Используется для отображения нескольких строк статического текста, обеспечивает поддержку простых и сложных стилизаций текста, а также возможность управления аспектами появления текста (добавление тени, изменение цвета и т.д.).
UIPickerView	Реализует объекты, предназначенные для просмотра на специальном элементе (визуально напоминающий «слот-машину» или «прялку»), чтобы показать несколько наборов значений. Пользователь выбирает значения при вращении колеса так, чтобы желаемая строка значений совпала с индикатором выбора.
UIProgressView	Используется для отображения хода выполнения задачи в течение длительного времени. Представляет свойства для управления стилем строки прогресса и получения и установки значений, которые отображают ходы выполнения операции.
UIActivityIndicatorView	Используется для отображения активности приложения, чтобы показать, что задача выполняется. Дает возможность управлять началом вызова, остановкой вызова, скрытием индикатора активности и изменением цвета индикатора.
UIImageView	Реализует объекты для просмотра изображений на основе контейнера для отображения либо одного изображения, либо серии изображений.
UITabBar	Панель вкладок, являющаяся элементом управления. Как правило, появляется в нижней части экрана в контексте контроллера панели вкладок для предоставления модального доступа к различным разделам приложения. Каждая кнопка в панели вкладок называется tab bar item и является экземпляром класса `UITabBarItem`. Класс `UITabBar` поддерживает пользовательские настройки панели вкладок по переназначению, удалению и добавлению элементов в баре. Присутствует возможность дополнительно анимировать поведение вкладок.
UIToolBar	Панель инструментов для управления одним или несколькими элементами кнопок, которые могут изменять внешний вид элементов на данном экране. Для создания панели инструментов используется класс `UIBarButtonItem`.
UINavigationBar	Обеспечивает управление для навигации иерархического содержа-

	ния. Обычно отображается в верхней части экрана, содержит кнопки для навигации иерархии экранов. Основные свойства разделены между тремя частями: левая часть – кнопка «Назад», центральная часть – название экрана, правая часть – дополнительные элементы управления. Данный класс можно использовать как самостоятельный объект либо в сочетании с объектом `UINavigationController`.
`UITableViewCell`	Определяет атрибуты и поведение ячеек, появляющихся в объектах класса `UITableView`. Данный класс включает свойства и методы для установки и управления контентом ячеек и фоном (включая текст, изображение, пользовательские представления), управления выбором ячейки, состояния выделения, инициирования редактирования содержимого ячейки.
`UIActionSheet`	Предназначен для представления пользователю альтернативных вариантов выполнения той или иной задачи. Также применяется для дополнительного подтверждения потенциально опасного действия. Лист действия содержит необязательный заголовок, одну или несколько кнопок, каждая из которых соответствует определенному действию. Методы и свойства данного класса предназначены для настройки сообщения, установки стиля кнопок, предоставления перечня действий. Для приложений, разработанных на iPhone/iPod Touch, данный объект появляется из нижней части экрана, а для приложений, разработанных для iPad, данный элемент появляется в другом окне (popover).
`UIAlertView`	Данный класс используется для отображения предупреждающих сообщений пользователю. Свойства и методы данного класса позволяют задавать заголовок, текст сообщения, настраивать кнопки, устанавливать делегат на вызываемый объект.
`UIScrollView`	Отвечает за отображение содержимого, размер которого больше размера окна приложения. Позволяет пользователям прокручивать содержимое на экране, изменять масштаб. Также данный класс является наследником для нескольких классов `UIKit`, включая `UITableView` и `UITextView`.
`UITableView`	Наследуется от `UIScrollView`, является средством для отображения и редактирования иерархического списка информации. Элемен-

	ты таблицы отображаются в одном столбце, доступна только вертикальная прокрутка таблицы.
UITextView	Наследуется от `UIScrollView`, предназначен для отображения многостраничной текстовой информации. Класс поддерживает возможность отображения пользовательской информации, а также редактирование текста.
UISearchBar	Предназначен для поиска по тексту. Состоит из поля для ввода текста, кнопки поиска, кнопки отмены.
UIWebView	Предназначен для отображения веб-контента в приложении. В использовании достаточно прост: объявляется данный класс и передается ссылка на веб-страницу. Присутствует возможность перемещения вперед и назад по истории веб-страниц. По умолчанию, данный объект преобразует телефонные номера, которые присутствуют на веб-странице для быстрого набора номера.
UIControl	Является базовым классом для объектов управления, которые передают действия пользователя в приложение (кнопки, ползунки и т.д.). Данный класс непосредственно использоваться не может, он определяет общий интерфейс и поведенческую структуру для всех его подклассов. Основная роль данного класса заключается в определении реализации интерфейса и базы для подготовки сообщения к действию и первоначальной отправки сообщения наследуемым объектам при наступлении данного события.
UIButton	Экземпляр класса реализует кнопку на сенсорном экране, кнопка перехватывает события и отправляет сообщения целевому объекту. Данный класс наследуется от `UIControl`.
UIDatePicker	Реализует объект, который использует несколько вращающих колес, предназначенных для выбора даты и времени пользователем. Данный класс наследуется от `UIControl`.
UIPageControl	Предназначен для создания и управления страниц в приложении. Для информирования пользователя о количестве страниц присутствует горизонтальный ряд точек, каждая из которых соответствует странице в документе. Текущая страница помечается белой точкой. Данный класс наследуется от `UIControl`.

UISegmentedControl	Данный объект состоит из горизонтально расположенного ряда кнопок, функционирующих в виде дискретных состояний. Данный класс наследуется от `UIControl`.
UITextField	Является элементов управления, отображает редактируемый текст и отправляет сообщение действий целевым объектам. Применяется для получения короткой информации от пользователя на основе текста. Данный класс наследуется от `UIControl`.
UISlider	Является элементом управления для выбора одного значения из непрерывного диапазона значений. Ползунки отображаются в виде горизонтальных баров. Данный класс наследуется от `UIControl`.
UISwitch	Используется для создания кнопки и ее– включения/выключения, например, для настроек приложения. Данные объекты известны как коммутаторы. Данный класс наследуется от `UIControl`.

Объекты `UIView` могут менять как свои размеры, так и размеры дочерних компонент при помощи следующих способов:

- `Autoresizing masks` – описывает относительное изменение размера и положения `UIView` относительно своего `superview` (см. рисунок ниже).

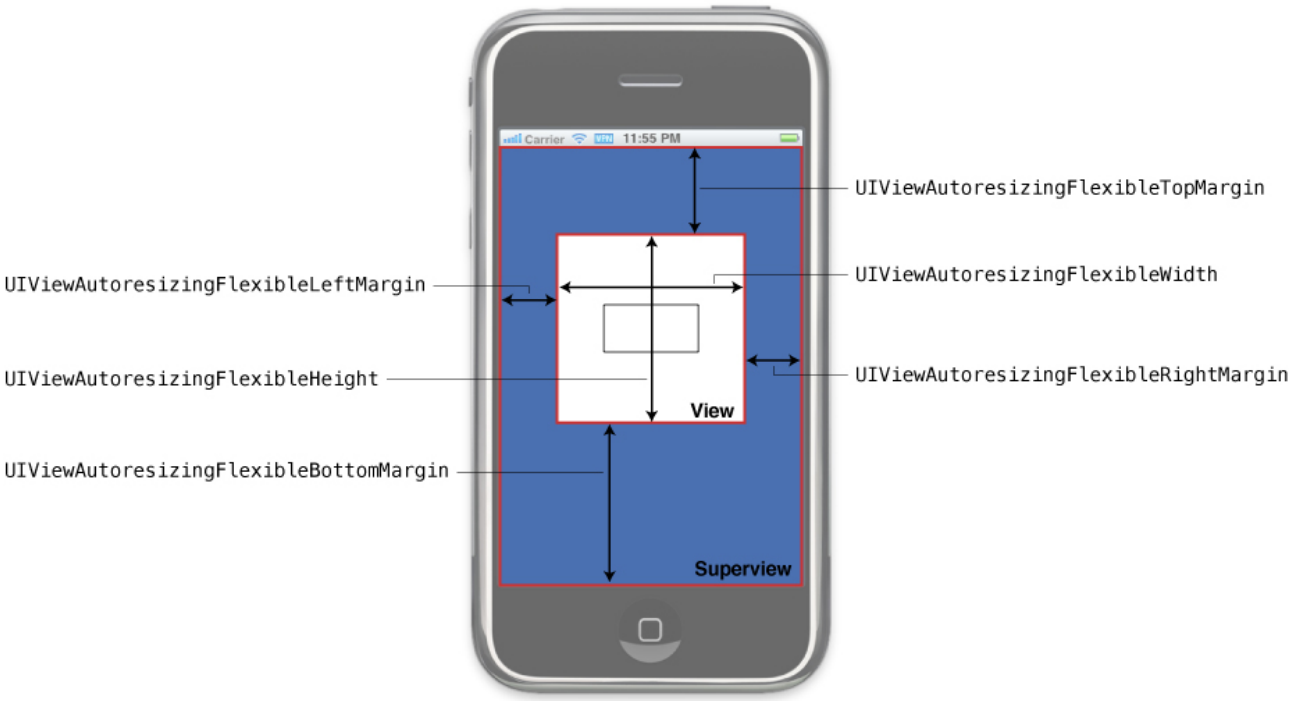


Рисунок 19. Константы Autoresizing mask

- `Autolayout` – описывает ограничения, относительно которых UI-компоненты изменяют свой размер при изменении размера своего контейнера (см. рисунок ниже).

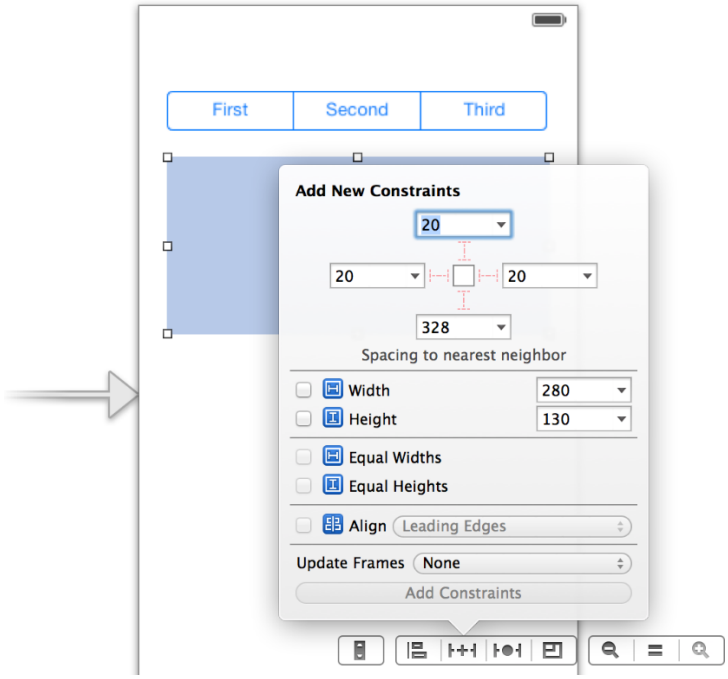


Рисунок 20. Параметры Autolayout

В iOS координатная система экрана разделена на два уровня:

1. **логический уровень** – измеряется в точках (points). Именно в этой системе и указываются размеры и положения UI-элементов;
2. **физический уровень** – определяет, сколько пикселей соответствует одной точке (pixels per point).

Таким образом, на устройствах с Retina дисплеем и на устройствах с обычным дисплеем размеры UI элементов, определенные в исходном коде, не изменяются. Класс `UIScreen` обладает свойством `scale`, которое возвращает количество пикселей в одной точке (для Retina дисплеев: `scale = 2.0`; для обычных дисплеев: `scale = 1.0`).

В UIKit началом координат является верхний левый угол, положение и размеры компонентов UI описываются **четырьмя свойствами** (см. рисунок ниже).

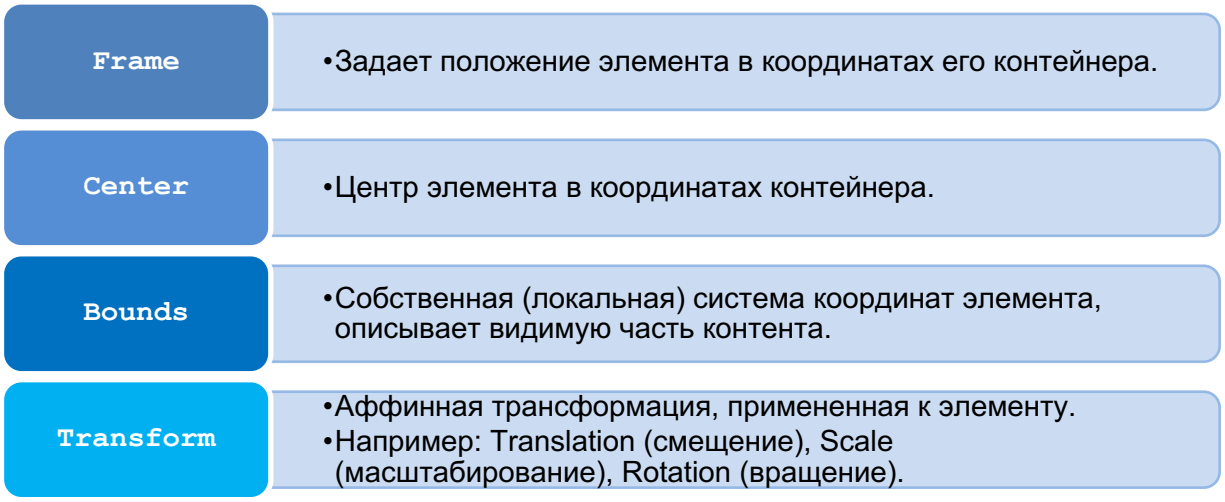


Рисунок 21. Свойства описывающие координаты объектов приложения в библиотеке UIKit

Каждый наследуемый класс от `UIView` имеет следующие свойства для управления иерархиями компонентов:

- `subviews` – объекты типа `UIView`, добавленные на текущий `UIView`;
- `superview` – `UIView`-контейнер, в котором содержится `UIView`.

Для удаления компоненты из формы можно использовать метод:

```
(void) removeFromSuperview
```

Помимо прочего, можно **управлять порядком отображения компонент:**

- `(void) bringSubviewToFront:(UIView*) view` – помещает UI объект по верх остальных UI объектов;
- `(void) sendSubviewToBack:(UIView*) view` – помещает UI объект в самых низ относительно других UI объектов;
- `(void) insertSubview:(UIView*) view atIndex:(NSInteger) index` – устанавливает UI объект по указанному индексу.

Класс `UIViewController` обеспечивает базовую модель управления представлениями в iOS согласно шаблону проектирования MVC (см. рисунок ниже). Контроллер представлений применяется, когда необходимо:

- изменять размеры и отображать представления;
- настраивать содержимое представлений;
- действовать от имени представлений, когда пользователь взаимодействует с ним.

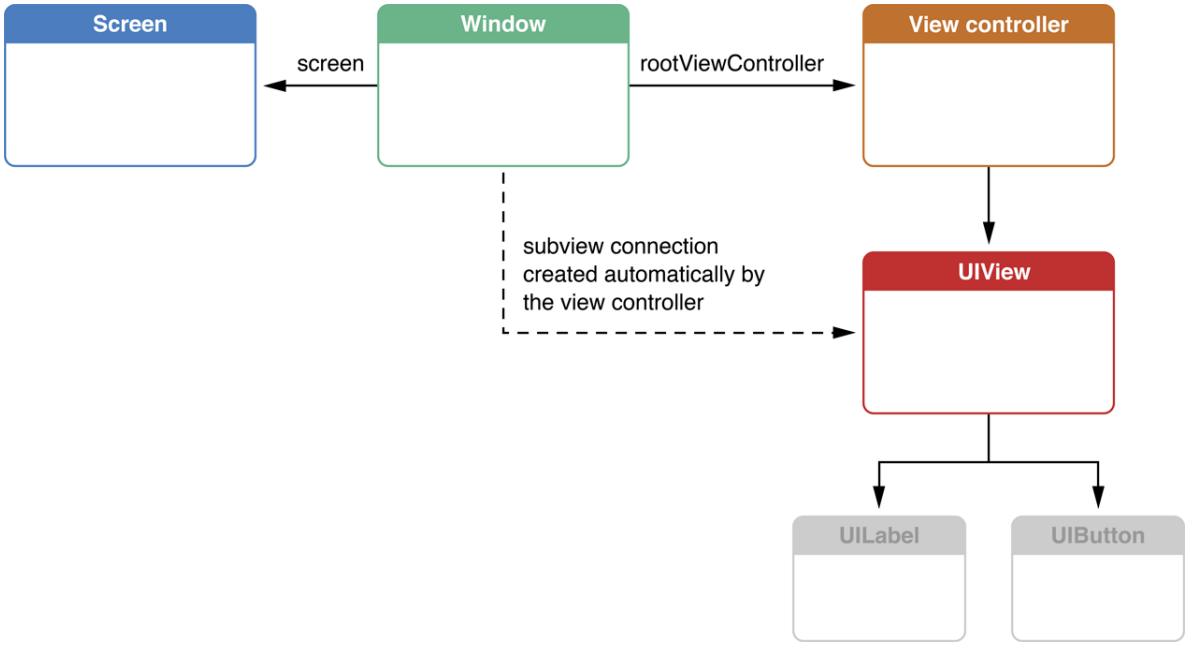


Рисунок 22. Иерархия приложения

Отдельно необходимо выделить контроллеры-контейнеры, используемые для иерархического или разделенного представления своих дочерних контроллеров представлений, к ним относятся:

- `UINavigationController` – системный контроллер, управляющий стэком контроллеров (в виде массива представлений). Обеспечивает навигацию и анимированные переходы между контроллерами представлений (чаще всего с таблицами, контроллер навигации обеспечивает переход от краткого отображения данных в таблице к детальному представлению элемента), используется для представления иерархической информации;
- `UITabBarController` – контроллер для закладок. Управляет переходом по закладкам, используется для представления разделенной информации. Каждая вкладка `UITabBarController` связана с отдельным контроллером представления;
- `UISplitViewController` – является раздельным контроллером представления. Предназначен только для приложений на iPad, реализует механизм работы с данными в формате «главный-подчиненный», представление контроллера слева реализует список элементов, а представление контроллера справа содержит сведения о выбранном элементе.

Табличные представления – наиболее распространенный механизм для отображения списков данных пользователю.

Табличная форма удобна для представления разнообразного типа данных, например, представления списка контактов, дел в календаре, плейлистов в плеере, альбомов фотографий и т.д. (см. рисунок ниже).

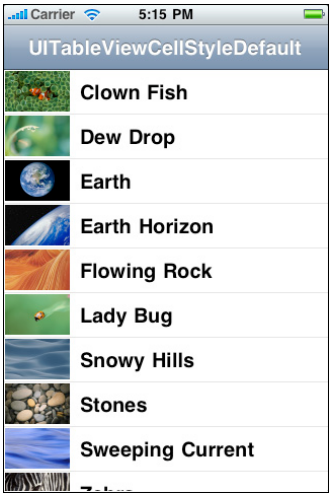


Рисунок 23. Табличное представление в iOS SDK

***Табличное представление** является объектом, который выводит различные данные, и представляет собой экземпляр класса `UITableView`, а каждая строка таблицы реализуется классом `UITableViewCell`.*

Таким образом, табличное представление является объектом, который выводит видимую часть таблицы, а ячейка табличного представления отвечает за отображение одной строки таблицы. Чаще всего таблицы используются вместе с `UINavigationController`, который обеспечивает навигацию между таблицей и детальным представлением элемента (см. рисунок ниже).

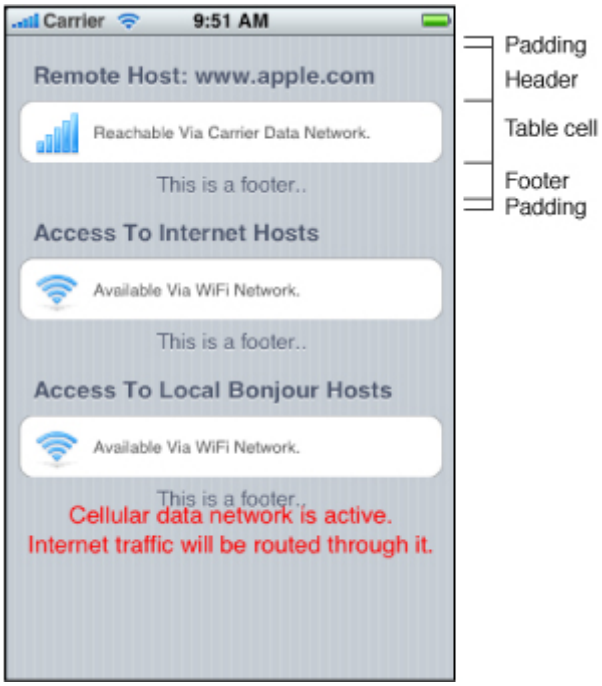


Рисунок 24. Структура табличного представления в iOS SDK

Табличные представления не несут ответственности за хранение табличных данных. Они хранят только те данные, которые необходимы для вывода строк, видимых в текущий момент времени. Табличные представления получают свои конфигурационные данные из объекта, соответствующего протоколу `UITableViewDelegate`, и строчные данные из объекта, соответствующего протоколу `UITableViewDataSource` (является моделью концепции MVC).

Дополнительно класс `UITableView` является гибкой надстройкой над `UIScrollView`, представляющая программный интерфейс манипулирования данными, таким образом можно использовать в работе делегат `UIScrollViewDelegate`. Каждый объект `UITableViewCell` может быть настроен для вывода изображения, некоторого текста и необязательной вспомогательной пиктограммы. При необходимости можно создать свое представление ячейки, добавив в объект класса `UITableViewCell` дочерние представления.

Процесс создания любого приложения можно условно разделить на три этапа (см. рисунок ниже).

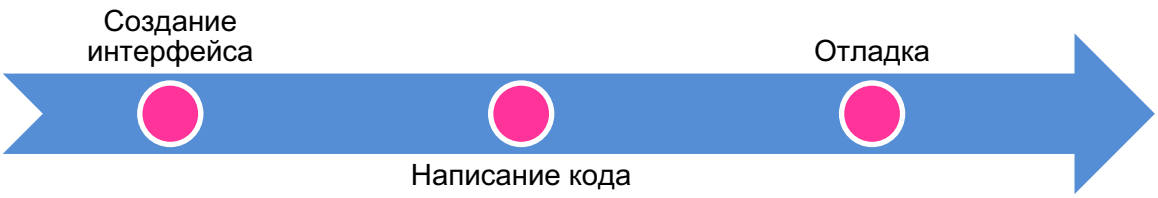


Рисунок 25. Процесс создания мобильного приложения

Одной из основных особенностей языка программирования Objective-C является общение при помощи отправки сообщений. Данная концепция нашла отражение при построении интерфейса – объекты интерфейса взаимодействуют между собой при помощи связей.

Связи бывают двух типов:

1. Связи типа `Outlet` представляют собой обычную переменную объекта, которая ссылается на объект в интерфейсе приложения. В коде приложения такие переменные классов отличаются от обычных спецификатором типа `IBOutlet`.
2. Связи типа `Action` необходимы для передачи сообщений от одного объекта к другому. Всякий раз, когда пользователь взаимодействует с интерфейсом программы (нажимает кнопки, перемещает слайдер, вводит текст в поле ввода) генерируются сообщения, обработку которых можно реализовать в коде.

При создании связей типа `Action` и `Outlet` разрешены соединения типа один-ко-многим.

В качестве редакторов интерфейса представлены инструменты *Interface Builder* и *Storyboard*, которые являются средствами для визуального создания и тестирования интерфейсов (входят в состав iOS SDK).

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.
2. Интерфейс приложения создать с использованием визуального редактора Stoyboard.

3. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
4. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
5. Установить связь между интерфейсом приложения и логикой приложения, в соответствие с которой происходит обработка данных.
6. Локализовать приложение на русский и английский язык.
7. Собрать проект и запустить на симуляторе.

Вариант 1

Создать приложение, состоящее из трех вкладок:

1. Первая вкладка содержит список элементов периодической таблицы Менделеева (можно не все элементы, а часть, по собственному выбору), представленных на основе таблицы (`UITableView`) с индексацией данных (перемещение по списку при помощи индекса). Ячейка должна содержать: название элемента, обозначение элемента, порядковый номер, атомную массу (расположение элементов в ячейки на усмотрение исполнителя данной работы). При нажатии на ячейку должен осуществляться переход на экран подробного описания элемента, сверстанного в виде отдельного представления (`UIView`) (дополнительно к вышеперечисленным полям добавить информацию: номер периода, номер группы).
2. Вторая вкладка содержит веб-страницу с сайтом – http://mti.edu.ru/. Предусмотреть возможность навигации по сайту (кнопки Вперед, Назад, Обновить, Отмена).
3. Третья вкладка содержит коллекцию изображений с возможностью открытия изображения на полный экран. В полноэкранном режиме должен быть реализован следующий функционал:

- масштабирование изображения при помощи жестов (Pinch Close, Pinch Open);
- переход к соседним изображениям при помощи листания.

Вариант 2

Создать приложение, состоящее из трех вкладок:

1. Первая вкладка содержит список столиц стран мира (можно не все 193 страны, а часть), представленных на основе таблицы (`UITableView`) с индексацией данных (перемещение по списку при помощи индекса) и поиска по ключевым словам (ввод первых букв – и сокращение до подходящих значений). Ячейка должна содержать: название столицы, название страны (расположение элементов в ячейки на усмотрение ис-

полнителя данной работы). При нажатии на ячейку должен осуществляться переход на экран с отображением столицы на карте (`MKMapView`).

2. Вторая вкладка содержит веб-страницу с сайтом http://mti.edu.ru/. Предусмотреть возможность навигации по сайту (кнопки Вперед, Назад, Обновить, Отмена).
3. Третья вкладка содержит коллекцию изображений с возможностью открытия изображения на полный экран. В полноэкранном режиме должен быть реализован следующий функционал:

- масштабирование изображения при помощи жестов (Pinch Close, Pinch Open);
- переход к соседним изображениям при помощи листания.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Что такое шаблон MVC? Каковы основные принципы, цели, назначение шаблона?
2. Что представляет собой система координат iOS? Каковы ее особенности, основные свойства?
3. Какими способами можно реализовать автоматическое изменение размеров объектов класса `UIView`?
4. Что представляет собой жизненный цикл контроллера представлений?
5. Каковы основные модальные контроллеры представлений и каково их назначение?
6. Какие контроллеры-контейнеры присутствуют в iOS SDK?
7. Какие основные стили есть у табличных представлений?
8. Что представляет собой структуру табличного представления?
9. Каковы способы оптимизации процессорного времени при создании ячеек (`UITableViewCell`) в табличном представлении?
10. Каковы отличия между визуальным редактором интерфейсов Storyboard и Interface Builder?
11. Каков процесс локализации приложения для iOS?

Литература

1. Дэйв Марк, Джек Наттинг, Джефф Ламарш. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch. Вильямс, 2012.

2. Стефан Кочан. Программирование на Objective-C 2.0. Издательство: ЭКОМ Паблишерз, 2010.
3. Ресурс разработчиков Apple.
4. Документация Apple по разработке и проектированию приложений для iOS.

Лабораторная работа № 5
«Работа с графикой и анимацией в iOS SDK»

Цель работы

1. Получить практические знания работы с библиотеками для работы с графикой и анимацией.
2. Изучить технологии Core Graphics, Core Animation.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Основы работы с анимацией и графикой в iOS SDK».

Теоретическая часть

В iOS SDK существует библиотека для работы с графикой и анимацией – Quartz 2D, входящая в состав оболочки Core Graphics.

Технология Core Graphics *представляет собой прикладные программные интерфейсы (API), написанные на языке программирования.*

Библиотека Quartz 2D *состоит из функций, типов данных и объектов, предоставляющих возможность рисовать непосредственно в представлении или изображении, находящемся в оперативной памяти.*

В Quatz 2D представление или изображение трактуется как «рисуемое на виртуальном холсте» и следует так называемой модели – painter's model (т.е. команды рисования применяются подобно накладываемой на холст краске).

Например: *если заполнить все представление сначала красным цветом, а затем нижнюю половину – синим цветом, то представление получится наполовину красным и наполовину синим. Если синий цвет будет полупрозрачным, то в нижней части представления получится фиолетовый цвет.*

Каждое рисующее действие накладывается на виртуальных холст поверх любых предыдущих рисующих действий (см. рисунок ниже).

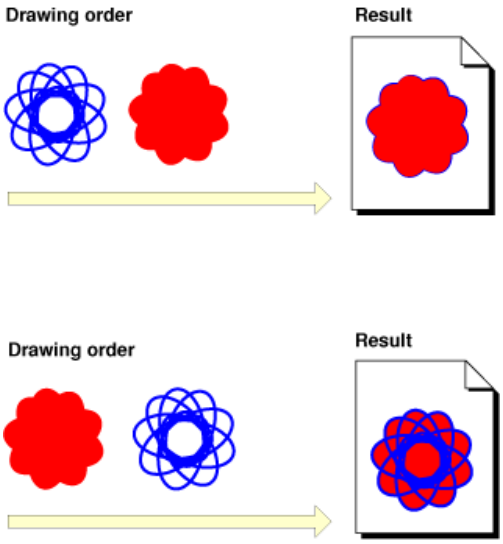


Рисунок 26. Принцип работы технологии Quartz 2D

В библиотеке Quartz 2D предоставляются разнообразные функции рисования линий, форм и изображений. Область действия технологии Quartz 2D ограничивается двумерной графикой. Во многих функциях библиотеки Quartz 2D используются преимущества аппаратного ускорения процесса рисования.

В библиотеке Quartz 2D, как и в библиотеке Core Graphics, рисование происходит в графическом контексте (graphics context). С каждым представлением связан определенный контекст, который извлекается и используется для вызова различных графических функций Quartz 2D. Основное предназначение контекста – визуализация рисуемой графики в представлении.

Для извлечения контекста используется следующее выражение:

```
CGContextRef context = UIGraphicsGetCurrentContext ();
```

После чего можно приступать к рисованию в представлении, что осуществляется путем передачи контекста различным графическим функциям оболочки Core Graphics.

Например: *для создания контура, состоящего из линий толщиной 3 пикселя, сначала создается контур, а затем уже рисуется линия:*

```
//Линия, из которой создается контур, должны быть толщиной 3 пикселя
CGContextSetLineWidth (context, 3.0);

//Линии контура должны быть синего цвета
```

```
CGContextSetStrokeColorWithColor (context, [UIColor blueColor].CGColor);

//Устанавливается начальная точка рисования
CGContextMoveToPoint (context, 10.0f, 10.0f);

//Устанавливается конечная точка рисования линии
CGContextAddLineToPoint (context, 20.0f, 20.0f);

//Отрисовка линии согласно указанным выше параметрам
CGContextStrokePath (context);
```

С контекстом связано невидимое перо, которым рисуется линия. По ходу выполнения команд рисования движениями данного пера образуется контур. При вызове метода `CGContextMoveToPont()` конечная точка текущего контура перемещается в заданное место, но при этом ничего не рисуется. Любая последующая операция будет выполнена относительной той точки, в которую перемещено перо.

Основные типы данных библиотеки Quartz 2D представлены на рисунке ниже.

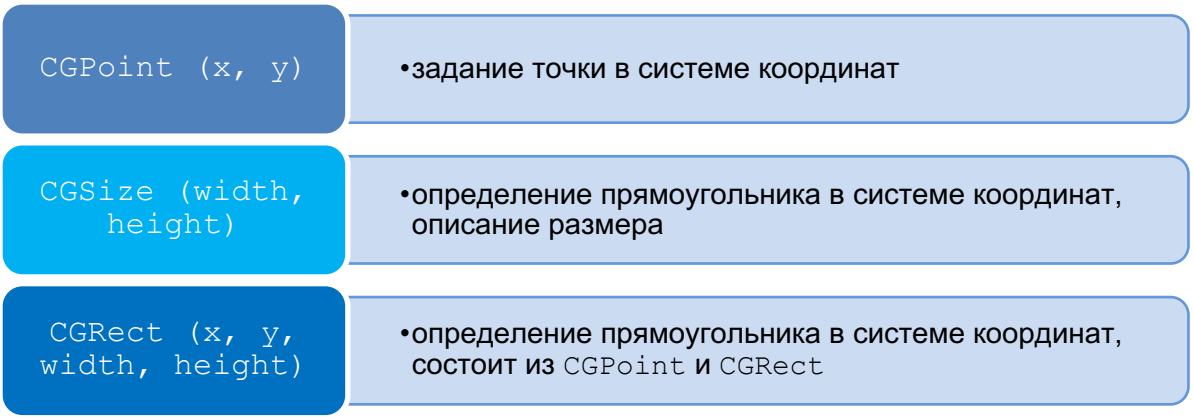


Рисунок 27. Основные типы данных библиотеки Quartz 2D

В iOS SDK распространено представление цвета как разложение на четыре составляющие: три составляющие RGB (красный, синий, зеленый) и альфа-канал. Каждая из составляющих представлена значением типа `CGFloat` в пределах от 0.0f до 1.0f. Альфа-канал определяет степень прозрачности цвета. При нанесении одного цвета на другой, альфа канал используется для формирования отображения окончательного цвета на экране. При значении альфа-канал, равному 1.0f, цвет получается абсолютно не прозрачным. Данная модель представления цвета называется **RGBA**.

Помимо RGBA в iOS SDK применяется и другие цветовые модели:

- HSV (Hue, Saturation, Value) – тон, насыщенность, значение;
- HSL (Hue, Saturation, Lightness) – тон, насыщенность, яркость;
- CMYK (Cyan, Magenta, Yellow, BlacK) – голубой, пурпурный, желтый, черный;

- Grayscale – представление изображения в оттенках серого.

Для работы с цветом в iOS SDK имеется класс `UIColor`, содержащий обширную группу методов, возвращающих объекты типа `UIColor`.

Для рисования в представлении в базовом классе `UIView` определен метод:

```
- (void) drawInRect:(CGRect)rect;
```

в котором можно производить отрисовку графического контента. Данный метод вызывается каждый раз, когда представлению требуется перерисовка.

Ни в коем случае не стоит вызывать данный метод самостоятельно, так как система автономно вызывает данный метод в нужный момент, а именно при:

- добавлении / удалении `UIView` в иерархию UI-компонентов;
- выставлении `UIView` свойства `hidden` в позицию `NO`;
- вызове метода `setNeedsDisplay` или `setNeedsDisplayInRect:`;
- добавлении / удалении другого объекта `UIView`, который перекроет текущий `UIView`.

При вызове метода `drawRect:` система создает контекст, в котором можно рисовать различные объекты. При необходимости, можно создавать свои контексты и рисовать в них изображения, градиенты и т.д.

```
- (void) drawRect:(CGRect) rect
{
CGContextRef context = UIGraphicsGetCurrentContext();
CGContextSetFillColorWithColor (context, self.backgroundColor.CGColor);
CGContextFillRect (context, rect);
}
```

Core Animation представляет собой набор Objective-C классов для рендеринга графики, проектирования и анимации. Core Animation обеспечивает изменение анимации с использованием передовых эффектов композиции, сохраняя иерархический уровень абстракции.

Неявная модель анимации Core Animation предполагает, что все изменения в анимируемых свойствах слоя должны быть постепенными и асинхронными. Например, неявная анимация свойства позиции слоя:

```
// Предположим, что текущее положение слоя в (100.0,100.0)
theLayer.position=CGPointMake(500.0,500.0);

// анимация theLayer's непрозрачность до 0 при отдалении в слое
theLayer.opacity=0.0;
theLayer.zPosition=-100;

// анимация anotherLayer непрозрачность до 1 при приближении в слое
```

```
anotherLayer.opacity=1.0;
anotherLayer.zPosition=100.0;
```

Неявные анимации используют длительность действия, указанную для свойства анимации по умолчанию, если длительность не была переопределена в явной или неявной транзакции.

Core Animation также поддерживает модель **явной анимации**.

Явная модель анимации *требует создания объекта анимации и установления начальных и конечных значений.*

Явная анимация не начнется, пока не будет применена анимация слоя.

```
CABasicAnimation *theAnimation;
theAnimation=[CABasicAnimation animationWithKeyPath:@"opacity"];
theAnimation.duration=3.0;
theAnimation.repeatCount=2;
theAnimation.autoreverses=YES;
theAnimation.fromValue=[NSNumber numberWithFloat:1.0];
theAnimation.toValue=[NSNumber numberWithFloat:0.0];
[theLayer addAnimation:theAnimation forKey:@"animateOpacity"];
```

Явная анимация начинается с отправки сообщения `addAnimation:forKey:` в целевой слой, которое передает анимацию и идентификатор в качестве параметра. После добавления к целевому слою явная анимация будет работать до завершения анимации либо удаления из слоя. Идентификатор используется для добавления анимации к слою, а также для ее остановки, ссылаясь на `removeAnimationForKey:`. Также можно остановить все анимации для слоя, отправив слою сообщение `removeAllAnimations`.

Начиная с iOS 4, представлены следующие методы анимации класса `UIView`:

```
+ (void)animateWithDuration:(NSTimeInterval)duration
                      delay:(NSTimeInterval)delay
                    options:(UIViewAnimationOptions)options
                 animations:(void (^)(void))animations
                 completion:(void (^)(BOOL finished))completion;
+ (void)animateWithDuration:(NSTimeInterval)duration
                 animations:(void (^)(void))animations
                 completion:(void (^)(BOOL finished))completion;
+ (void)animateWithDuration:(NSTimeInterval)duration
                 animations:(void (^)(void))animations;
+ (void)transitionWithView:(UIView *)view
                  duration:(NSTimeInterval)duration
                   options:(UIViewAnimationOptions)options
                animations:(void (^)(void))animations
                completion:(void (^)(BOOL finished))completion;
+ (void)transitionFromView:(UIView *)fromView
```

```
                toView:(UIView *)toView
              duration:(NSTimeInterval)duration
               options:(UIViewAnimationOptions)options
            completion:(void (^)(BOOL finished))completion;
```

В примере указанном выше представлены две основных групп методов – анимация изменения представлений, и анимация замены одного представления другим.

Первые три метода отличаются только набором параметров (см. рисунок ниже). Первый метод – наиболее подробный метод, третий метод – самый простой, так как большая часть параметров не задается.

Рисунок 28. Параметры группы методов +animatedWith…

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.
2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.

4. Установить связь между интерфейсом приложения и логикой приложения, в соответствие с которой происходит обработка данных.
5. Собрать проект и запустить на симуляторе.

Вариант 1

Создать приложение, задача которого заключается в построении линейного двумерного графика из заданного набора значений (см. рисунок ниже). На рисунке представлен пример конечного вида модуля построения графиков.

Рисунок 29. Пример построения линейного графика

Должны быть предоставлены следующие наборы значений для выбора пользователю:

- построение функции по формуле: $\frac{\sin x}{x}$ (x = -10 … 10);
- построение функции по формуле: x^2;
- построение функции по формуле: $\sqrt{x} \cdot cos(200 \cdot x) + \left(\sqrt{|x|} - 0.7\right) \cdot (4 - x^2)^{0.01}$;
- построение функции из массива набора чисел (числа для оси абсцисс и оси ординат создать случайным способом в диапазоне от -5 до 5, не менее 50 значений).

Вариант 2

Создать приложение, задача которого заключается в построении круговой диаграммы из заданного набора значений (см. рисунок ниже). На рисунке представлен пример конечного вида модуля построения графиков.

Рисунок 30. Пример построения круговой диаграммы

Набор значений для диаграммы должны быть сгенерированы случайным образом в диапазоне от 0 до 1000, не более 10 значений.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Какой принцип работы технологии Quartz 2D?
2. Каковы основные типы данных библиотеки Quartz 2D?
3. Какие цветовые модели применяются в iOS SDK?
4. Что такое неявная и явная модель анимации в рамках технологии Core Animation?

Литература

1. Nick Lockwood. iOS Core Animation: Advanced Techniques. Addison Wesley, 2013.

2. Дэйв Марк, Джек Наттинг, Джефф Ламарш. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch. Вильямс, 2012.
3. Ресурс разработчиков Apple.
4. Документация Apple по работе с технологией Core Animation.
5. Документация Apple по работе с библиотекой Quartz 2D.

Лабораторная работа № 6

«Работа с многопоточностью в iOS SDK»

Цель работы

1. Познакомиться с механизмами распараллеливания задач, представленных в iOS SDK.
2. Изучить работу класса NSOperation и NSOperationQueue.
3. Научиться работать с методом – `(void)performSelectorInBackground:(SEL)aSelector withObject:(id)arg`.
4. Получить практические знания работы с Grand Central Dispatch.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Многопоточность, очередь операций, Grand Central Dispatch».

Теоретическая часть

В многопоточных приложениях *обслуживание интерфейса производится в отдельном потоке, а обработка данных – в другом (одном или нескольких) потоке, что позволяет приложению сохранять возможность откликаться на действия пользователя даже во время интенсивной обработки данных.*

В языке программирования Objective-C предусмотрено несколько технологий для создания и управления потоками (см. рисунок ниже).

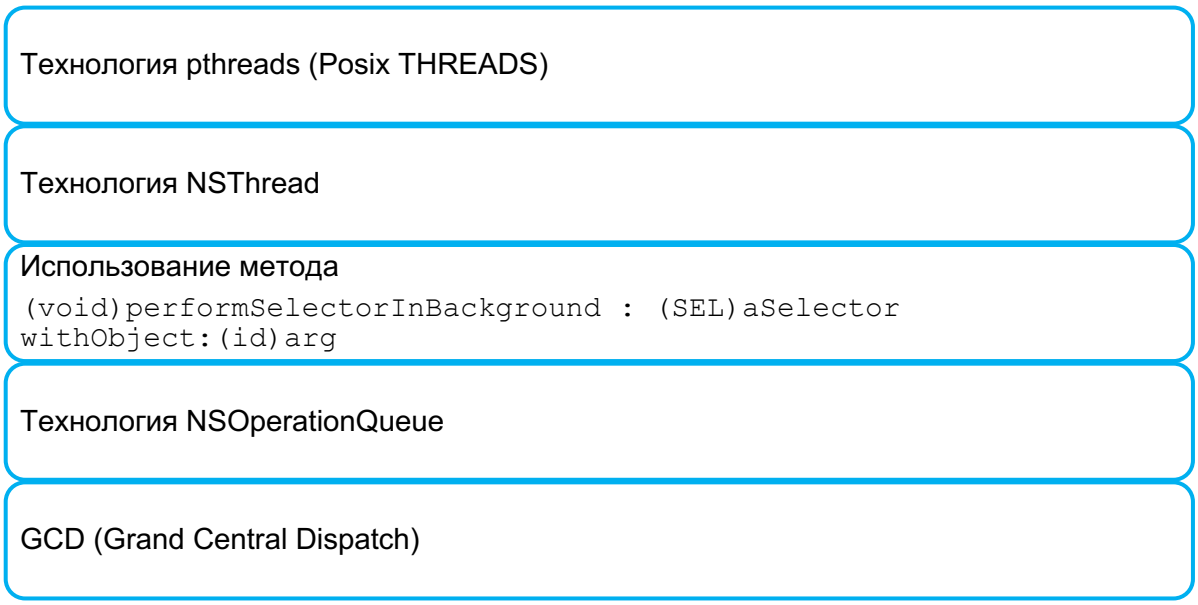


Рисунок 31. Технологии создания и управления потоками

- **Технология pthreads (Posix THREADS)** является кроссплатформенной технологией и на данный момент считается технологически устаревшей и сложной в реализации. Использование его актуально только для поддержки старых и портирования уже написанных приложений.
- **Технология NSThread** предполагает ручное создание потоков и управление их жизненным циклом; является объектной надстройкой над Posix-потоками (см. пример ниже). На данный момент технология NSThread не используется.

```
[NSThread detachNewThreadSelector:@selector(someAction:) toTarget:self
withObject:nil];

-(void) someAction:(id)anObject
{
    NSAutoreleasePool *autoreleasepool = [[NSAutoreleasePool alloc] in-
it];

    // исполняемый код

    [NSThread exit];
    // защита от утечек памяти
    [autoreleasepool release];
}
```

- **Использование метода**
  (void)performSelectorInBackground : (SEL)aSelector withOb-
  ject:(id)arg

является самым простым и распространенным методом обработки информации вне родительского потока. При реализации метода необходимо указать лишь селектор и его аргументы:

```
NSObject *object = [NSObject new];
[object    performSelectorInBackground:@selector(someAction:)    withOb-
ject:nil];

-(void) someAction:(id)anObject
 {
      @autoreleasepool {
      // исполняемый код
      }
}
```

- **Технология NSOperationQueue** является новым методом создания и управления потоками (см. пример ниже). Создается объект `NSOperation`, который выполняет новую задачу и по результату выполнения помещает его в очередь `NSOperationQueue`, которая затем выполняет все операции в очереди согласно заданным приоритетам.

```
NSOperationQueue *queue = [NSOperationQueue new];
queue.maxConcurrentOperationCount = 2;
NSOperation *objectOne = ..., *objectTwo = ..., *objectTrhee = ...;
[objectTrhee addDependency:objectTwo];
[queue addOperation:objectOne];
[queue addOperation:objectTwo];
[queue addOperation:objectTrhee];
```

- **GCD (Grand Central Dispatch)** является механизмом распараллеливания задач, при котором реализация многопоточности скрывается от программиста (представлен в iOS 4 и Mac OS X 10.6).

Диспетчер очередей позволяет выполнять произвольный блок кода, как асинхронно, так и синхронно по отношению к вызвавшему потоку.

*Механизмом, помогающим разработчику реализовать диспетчер очередей в приложениях iOS, является **Grand Central Dispatch** (GCD).*

GCD достаточно прост в использовании и имеет большую эффективность при выполнении задач по сравнении с ручной реализацией многопоточности с использованием низкоуровневых функций.

Операционная система обеспечивает для каждого приложения **четыре типа параллельных очередей:**

1. `DISPATCH_QUEUE_PRIORITY_DEFAULT` – осуществляет доступ к очереди с обычным приоритетом;
2. `DISPATCH_QUEUE_PRIORITY_HIGH` – осуществляет доступ к очереди с высоким приоритетом;
3. `DISPATCH_QUEUE_PRIORITY_LOW` – осуществляет доступ к очереди с низким приоритетом;
4. `DISPATCH_QUEUE_PRIORITY_BACKGROUND` – осуществляет доступ к очереди, которая работает в фоновом режиме.

Эти очереди являются глобальными и различаются только уровнем приоритета. Поскольку они носят глобальный характер, соответственно невозможно их создать в явном виде, вместо этого отправляется сообщение одной из очередей при помощи функции `dispatch_get_global_queue`, как показано в следующем примере:

```
dispatch_queue_t aQueue = dispatch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
```

Grand Central Dispatch предоставляет функции, которые позволяют получать доступ из приложения к нескольким общим очередям (см. рисунок ниже).

Функция dispatch_get_current_queue

- Используется для проверки назначения или идентичности очереди.
- Функция вызывается внутри блокового объекта и возвращается в очередь, в которой блок был представлен. Вызов данной функции за пределами блока возвращает параллельную очередь по умолчанию для данного приложения.

Функция dispatch_get_main_queue

- Используется для получения последовательной очереди отправки, связанной с основным потоком приложения. Эта очередь создается автоматически для Cocoa Touch-приложений и для приложений, которые либо вызвали функцию `dispatch_main`, либо настроили цикл выполнения (с помощью объекта типа `CFRunLoopRef` или `NSRunLoop`) в главном потоке.

Функция dispatch_get_global_queue

- Используется для получения любой из доступных параллельных очередей.

Функция dispatch_set_finalizer_f

- Используется для указания функции, которая будет выполняться, когда количество ссылок на данную очередь дойдет до нуля. Можно использовать данную функцию для очистки контекстных данных, связанных с очередью.

Рисунок 32. Функции Grand Central Dispatch

```
void myFinalizerFunction(void *context)
{
    MyDataContext* theData = (MyDataContext*)context;

    // Clean up the contents of the structure
    myCleanUpDataContextFunction(theData);

    // Now release the structure itself.
    free(theData);
}

dispatch_queue_t createMyQueue()
{
    MyDataContext*          data      =      (MyDataContext*)      mal-
loc(sizeof(MyDataContext));
    myInitializeDataContextFunction(data);

    // Create the queue and set the context data.
    dispatch_queue_t            serialQueue            =            dis-
patch_queue_create("com.example.CriticalTaskQueue", NULL);
    if (serialQueue)
    {
        dispatch_set_context(serialQueue, data);
        dispatch_set_finalizer_f(serialQueue, &myFinalizerFunction);
```

```
    }

    return serialQueue;
}
```

Функция `dispatch_once` используется, когда конкретную задачу нужно выполнить только один раз за все время жизни приложения. Это может быть полезным, например, при инициализации синглтона:

```
static dispatch_once_t task;
dispatch_once(&task, ^{
    // код
});
```

Для отложенного исполнения задачи на некоторое время следует использовать функцию `dispatch_after`. Указание времени, периода, через который необходимо запускать задачу на выполнение, представлено структурой `dispatch_time_t`, которая создается функцией `dispatch_time`. Функция определяет, сколько времени (второй аргумент) прошло от конкретного времени (первый аргумент).

Например: *для указания текущего времени можно использовать переменную* `DISPATCH_TIME_NOW`*. Время отсчета задается в наносекундах.*

Для запуска задачи на выполнение несколько раз следует использовать функцию `dispatch_apply`, которая возвращает управление по завершении всех запущенных задач вызываемому объекту.

```
dispatch_queue_t                queue                =                dis-
patch_get_global_queue(DISPATCH_QUEUE_PRIORITY_DEFAULT, 0);
dispatch_apply(10, queue, ^(size_t index) {
   // блок выполнится 10 раз
   // в блок передается номер запуска
});
```

Очередь операций *(Operation Queues) – объектно-ориентированный способ решения задач, которые должны выполняться асинхронно относительно главного процесса.*

Очередь операций предназначена для использования либо в сочетании с Dispatch Queues, либо самостоятельно.

Operation Queues является экземпляром класса `NSOperation` в библиотеке Foundation framework. Класс `NSOperation` содержит достаточное количество свойств и методов, кото-

рые позволяют минимизировать объем написанного кода при создании многопоточного приложения. Определение простого объекта операции показано в примере ниже:

```
@interface MyNonConcurrentOperation : NSOperation
@property id (strong) myData;
-(id)initWithData:(id)data;
@end

@implementation MyNonConcurrentOperation
- (id)initWithData:(id)data {
   if (self = [super init])
      myData = data;
   return self;
}

-(void)main {
   @try {
      // Do some work on myData and report the results.
   }
   @catch(...) {
      // Do not rethrow exceptions.
   }
}
@end
```

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.
2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Установить связь между интерфейсом приложения и логикой приложения, в соответствие с которой происходит обработка данных.
5. Собрать проект и запустить на симуляторе.

Вариант 1

1. Создать экран в приложении, состоящий из табличного представления (`UITableView`).
2. Ячейки таблицы должны содержать изображение и текстовую метку.

3. Текстовая метка заполняется на усмотрение исполнителя (можно предложить любой свой вариант, либо выводить индекс ячейки).
4. Изображение в ячейке должно загружаться по ссылке (например, http://mti.edu.ru/share/images/mti(1).gif, или можно использовать любое другое изображение).
5. Реализовать загрузку изображения в отдельном потоке следующими способами:
- при помощи метода – `(void)performSelectorInBackground: (SEL)aSelector withObject:(id)arg;`
- при помощи класса `NSOperationQueue`;
- при помощи технологии Grand Central Dispatch.
6. Проверить работу приложения на симуляторе.

Вариант 2

1. Создать экран в приложении, состоящий из представления «Коллекция» (`UICollectionView`).
2. Ячейки коллекции должны содержать изображение и текстовую метку.
3. Текстовая метка заполняется на усмотрение исполнителя (можно предложить любой свой вариант, либо выводить индекс ячейки).
4. Изображение в ячейке должно загружаться по ссылке (например, http://mti.edu.ru/share/images/mti(1).gif, или можно использовать любое другое изображение).
5. Реализовать загрузку изображения в отдельном потоке следующими способами:
- при помощи метода – `(void)performSelectorInBackground: (SEL)aSelector withObject:(id)arg;`
- при помощи класса `NSOperationQueue`;
- при помощи технологии Grand Central Dispatch.
6. Проверить работу приложения на симуляторе.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Каковы технологии Objective-C,используемые для создания и управления потоками?
2. Каковы основные особенности работы с GCD?
3. Каков процесс реализации многопоточного приложения с использованием классов `NSOperation` и `NSOperationQueue`?

Литература

1. Дэйв Марк, Джек Наттинг, Джефф Ламарш. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch. Вильямс, 2012.
2. Ресурс разработчиков Apple.
3. Документация Apple по созданию и управлению дополнительных потоков в приложении.

Лабораторная работа № 7

«Изучение фреймворка Core Data»

Цель работы

1. Познакомиться с фреймворком Core Data.
2. Получить практические знания по работе с механизмом взаимодействия приложения и хранилища данных.
3. Научиться создавать модели данных.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Сохранение данных».

Теоретическая часть

Сохранение данных (Persistence) является составной частью большинства iOS-приложений. К сохранению данных принято относить как сохранение настроек пользователя, так и управление большими наборами данных в реляционных базах данных.

Для сохранения данных на iOS-устройстве существуют различные механизмы, обеспечивающие сохранение данных в файловой системе (см. рисунок ниже).

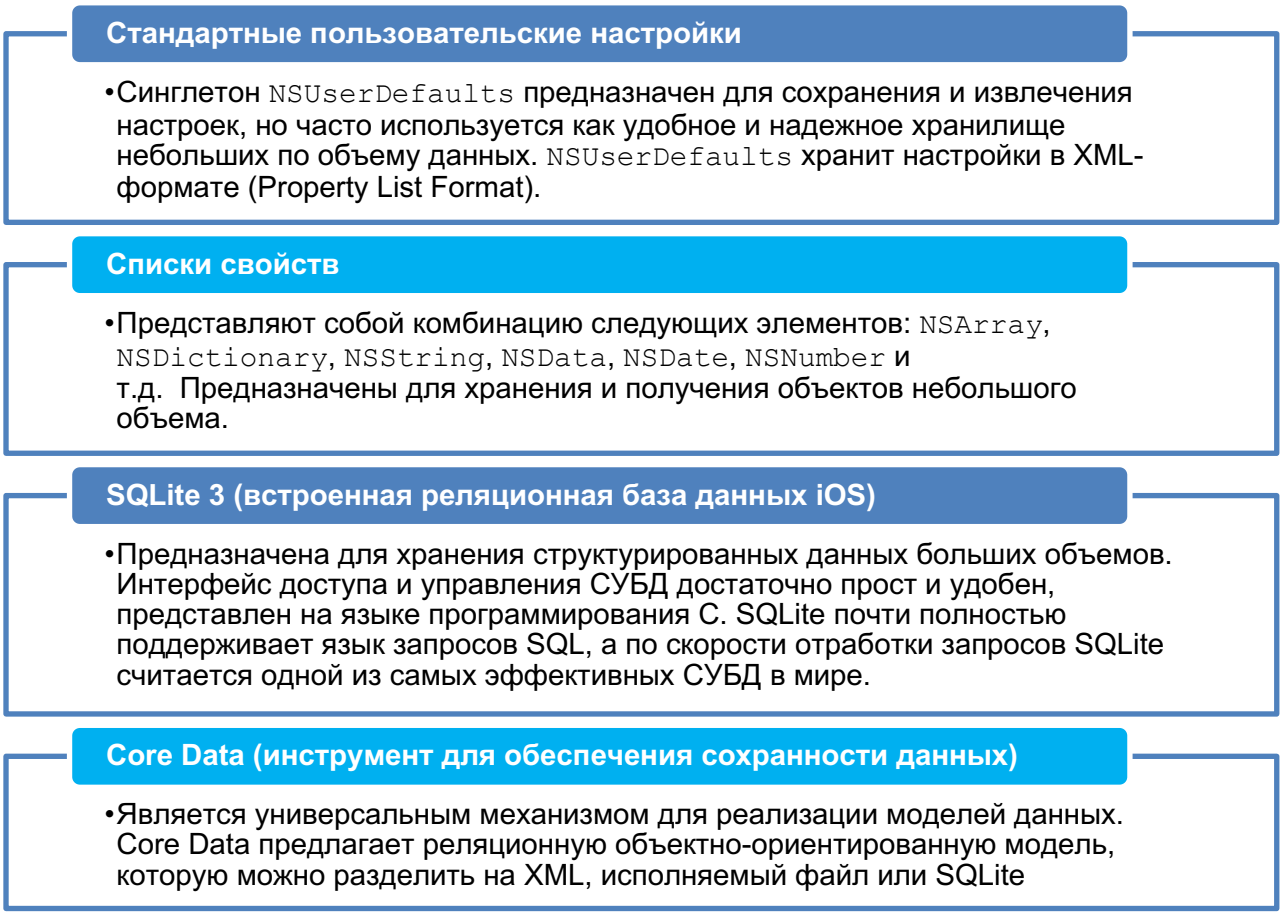


Рисунок 33. Механизмы сохранения данный приложения на iOS-устройстве

Все вышеперечисленные механизмы обеспечения персистентности данных используют один общий элемент – у каждого приложения есть собственная директория, и приложению при чтении и записи разрешается использовать только ее.

Внутри корневой директории находится три поддерживающие папки:

1. **Папка Documents** – используется для сохранения пользовательских данных, за исключением настроек, обслуживаемых на базе класса `NSUserDefaults`.

   Ниже приведен один из способов доступа приложения к директории Documents.

```
- (NSString *)applicationDocumentsDirectory
{
NSArray *paths = NSSearchPathForDirectoriesInDomains(NSDocumentDirectory,
NSUserDomainMask, YES);
NSString *basePath = ([paths count] > 0) ? [paths objectAtIndex:0] : nil;
return basePath;
}
```

Метод возвращает путь к директории Documents. С помощью методов классов `NSFileManager` и `NSString` можно создавать и удалять поддиректории этой директории, проверять файл на предмет существования, а также просматривать его файловые параметры.

Начиная с iOS 3.0, стало возможным получить доступ к директории Documents при помощи функции `NSHomeDirectory()`. Функция возвращает абсолютный путь к корневой директории, которая доступна приложению для работы с данными в файловой системе.

```
NSString* docsDir = NSHomeDirectory();
docsDir = [docsDir stringByAppendingPathComponent:@"Documents"];
```

2. **Папка Library** – предназначена для хранения данных, которые не обязательно связаны с пользователем, а также для хранения настроек, обслуживаемых на базе класса `NSUserDefaults`.
3. **Папка tmp** – предназначена для хранения временных файлов. Файлы, содержащиеся в данной папке, не подлежат синхронизации и резервному копированию, но в приложении должен быть реализован механизм, который будет отвечать за удаление ненужных файлов в данной директории. Для получения доступа к данной директории достаточно воспользоваться функцией `NSTemporaryDirectory()` из библиотеки Foundation, которая возвращает строку, содержащую полный путь к каталогу:

```
NSString *tempPath = NSTemporaryDirectory();
NSString *tempFile = [tempPath stringByAppendingPathCompo-
nent:@”tempFile.txt”];
```

Core Data является гибким фреймворком для работы с хранимыми на устройстве данными. Большинство деталей по работе с хранилищем данных Core Data скрывает от разработчика, позволяя ему сконцентрироваться на создании продукта.

Несмотря на то, что Core Data может хранить данные в реляционной базе данных (вроде SQLite), Core Data не является СУБД. Core Data в качестве хранилища может вообще не использовать реляционные базы данных.

> ***Core Data** является оболочкой/фреймворком для работы с данными, позволяющей работать с сущностями и их связями (отношениями к другим объектами), атрибутами в виде, напоминающем работы с объектным графом в обычном объектно-ориентированном программировании.*

При использовании Core Data никогда не нужно работать напрямую с хранилищем данных. Необходимо абстрагироваться от хранилища, от типа хранилища, необходимо мыслить в категориях, данных. Особенностью такого подхода является возможность безболезненно сменить тип хранилища (например, с XML файла, на SQLite) без изменения ранее написанного кода.

На рисунке ниже продемонстрированы основные компоненты библиотеки Core Data.

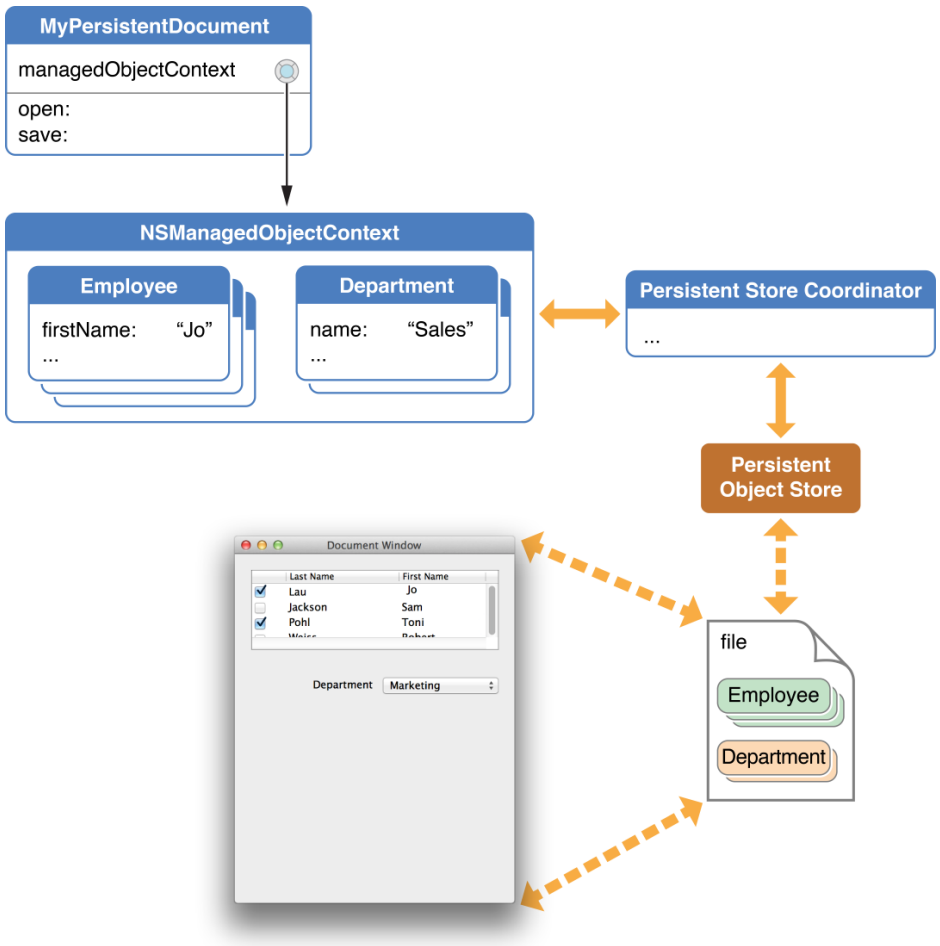


Рисунок 34. Основные компоненты Core Data

Объекты, которые находятся под управлением фреймворка (Core Data), должны наследовать методы/свойства класса `NSManagedObject`. Объекты располагаются в среде, которая называется **managed object context** (среда управляемых объектов) или просто **context**. Среда, в которой находится объект, следит не только за тем, в каком состоянии находится объект, но и за состояниями связанных объектов (объектов, которые зависимы от данного и от которых зависим он сам). Экземпляр класса `NSManagedObjectContext` предоставляет данную среду для объектов, объект данного типа должен быть доступен в приложении всегда. Обычно экземпляр класса `NSManagedObjectContext` является свойством делегата приложения.

Координатору постоянного хранилища (persistent store coordinator) отводится важная роль в управлении данными. Координатор используется для управления набором постоянных хранилищ, но обычно с ним не часто взаимодействуют разработчики, так как в типичном iOS-приложении используется только одно хранилище.

Файл Core Data имеет расширение *.xcdatamodeld – обычно к нему ссылаются как к управляемой модели объекта (managed object model) или Core Data model файлу. Чтобы добавить его в проект, необходимо выбрать в верхнем меню «File» → «New» → «New File» – и в разделе iOS → Core Data выбрать шаблон «Data Model».

В файл AppDelegate необходимо будет добавить **три свойства:**

```
@property (readonly, strong, nonatomic) NSManagedObjectContext
*managedObjectContext;
@property (readonly, strong, nonatomic) NSManagedObjectModel
*managedObjectModel;
@property (readonly, strong, nonatomic) NSPersistentStoreCoordinator
*persistentStoreCoordinator;
```

и два метода:

```
- (void)saveContext; //метод, сохраняющий изменения объектов модели в
файл данных
- (NSURL *)applicationDocumentsDirectory; //метод, указывающий на папку,
в которой содержатся файлы с данными
```

Для извлечения или выборки данных необходимо использовать классы:

- `NSFetchedResultsController` представляет собой контроллер, предоставляемый фреймворком Core Data для управления запросами к хранилищу.
- `NSManagedObjectContext` является описанной выше средой существования объектов типа `NSManagedObject`.

На рисунке ниже представлены основные шаги для получения данных из базы данных при помощи фреймворка Core Data.

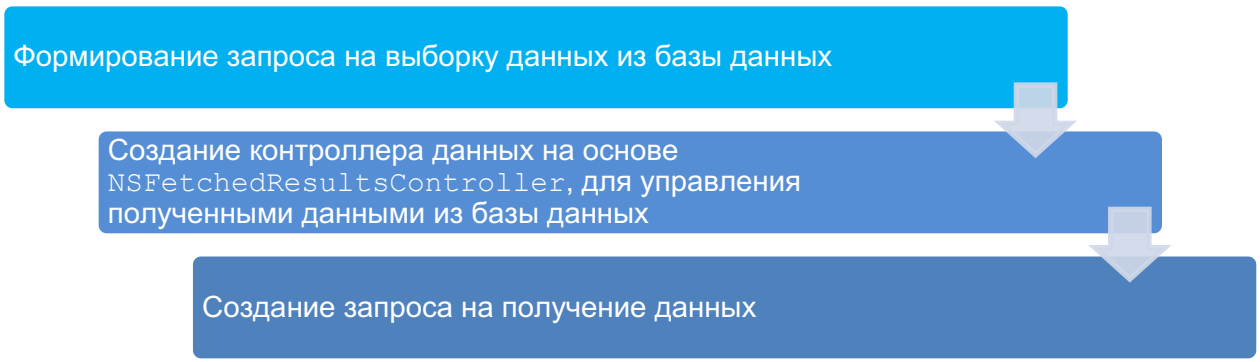


Рисунок 35. Шаги для выборки данных из хранилища данных при помощи библиотеки Core Data

Первым шагом к осуществлению запроса на выборку данных является создание запроса на выборку данных из базы данных:

```
NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
NSEntityDescription *entity = [NSEntityDescription entityForName:@"Event"
inManagedObjectContext:self.managedObjectContext];
[fetchRequest setEntity:entity];
```

Результаты запроса могут быть отсортированы при помощи `NSSortDescriptor`. `NSSortDescriptor` – определяет поле для сортировки и тип сортировки (по возрастанию или убыванию).

```
NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc] initWith-
Key:@"timeStamp" ascending:NO];
NSArray *sortDescriptors = [[NSArray alloc] initWithOb-
jects:sortDescriptor, nil];
[fetchRequest setSortDescriptors:sortDescriptors];
```

После того, как запрос определен, можно приступить к созданию `NSFetchedResultsController`. Используя в качестве делегата метод `NSFetchedResultsController`, можно следить за состоянием данных хранилища (удаление, добавление, перемещение и т.д.) и интегрировать данное решение с `UITableView`. Создание и настройка переменной экземпляра класса `NSFetchedResultsController`:

```
NSFetchedResultsController *aFetchedResultsController =
[[NSFetchedResultsController alloc] initWithFetchRequest:fetchRequest
managedObjectContext:self.managedObjectContext sectionNameKeyPath:nil
cacheName:@"Master"];
aFetchedResultsController.delegate = self;
self.fetchedResultsController = aFetchedResultsController;
```

В завершение остается только выполнить запрос для получения данных:

```
NSError *error = nil;
if(![self.fetchedResultsController performFetch:&error]){
    NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
    abort();
}
```

Для создания нового объекта необходимо выполнить ряд действий, показанных на рисунке ниже.

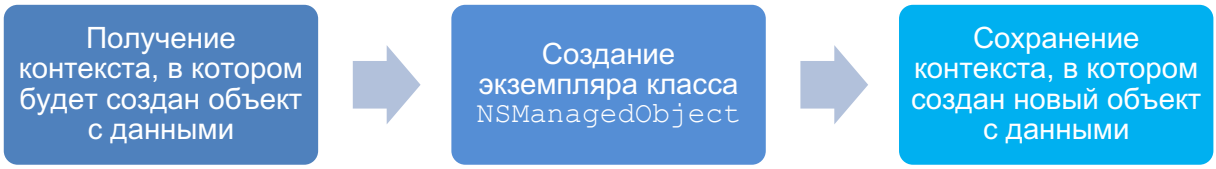


Рисунок 36. Описание процесса создания нового объекта для сохранения данных

при помощи библиотеки CoreData

На первом шаге необходимо получить контекст, в котором новый объект будет создан:

```
NSManagedObjectContext *managedObjectContext =
[self.fetchedResultsController managedObjectContext];
NSEntityDescription *entity = [[self.fetchedResultsController fetchRe-
quest] entity];
```

Затем необходимо создать экземпляр класса `NSManagedObject`:

```
NSManagedObject *newManagedObject = [NSEntityDescription insertNewObject-
ForEntity:[entity name] inManagedObjectContext:context];
[newManagedObject setValue:[NSDate date] forKey:@"timeStamp"];
```

Последним шагом, который необходимо осуществить, является сохранение контекста, в котором был создан новый объект. Но при этом стоит учитывать, что при сохранении контекста все несохраненные ранее изменения будут сохранены.

```
NSError *error = nil;
if(![context save:&error]){
   NSLog(@"Unresolved error: %@, %@", error, [error userInfo]);
   abort();
}
```

Для создания хранилища, поддерживающего созданные модели данных, можно использовать следующую конструкцию:

```
- (NSPersistentStoreCoordinator *)persistentStoreCoordinator {
   if(_persistentStoreCoordinator != nil){
      return _persistentStoreCoordinator;
   }

   NSURL *storeURL = [[self applicationDocumentsDirectory] URLByAppend-
ingPathComponent:@"BasicApplication.sqlite"];

   NSError* error = nil;
   _persistentStoreCoordinator = [[NSPersistentStoreCoordinator alloc]
initWithManagedObjectModel:[self managedObjectModel]];
   if(![_persistentStoreCoordinator addPer-
sistentStoreWithType:NSSQLiteStoreType configuration:nil URL:storeURL op-
tions:nil error:&error]){
      NSLog(@"Unresolved error %@, %@", error, [error userInfo]);
      abort();
   }

   return _persistentStoreCoordinator;
}
```

Для создания контекста (объекта, при помощи которого происходит взаимодействие с оболочкой Core Data) можно использовать следующий метод:

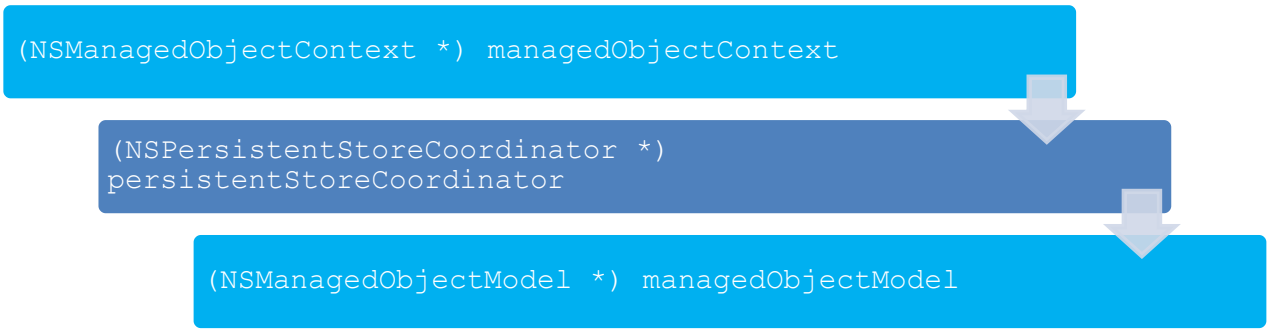
```
- (NSManagedObjectContext *)managedObjectContext {
    if(_managedObjectContext != nil){
        return _managedObjectContext;
    }

    NSPersistentStoreCoordinator *coordinator = [self persistentStoreCoor-
dinator];
    if(coordinator != nil){
        _managedObjectContext = [[NSManagedObjectContext alloc] init];
        [_managedObjectContext setPersistentStoreCoordinator:coordinator];
    }

    return _managedObjectContext;
}
```

Контекст используется во всем приложении в качестве интерфейса для взаимодействия с Core Data и постоянным хранилищем.

Таким образом, вызывая геттер свойства `managedObjectContext` делегата приложения на выполнение, запускается цепочка действий (см. рисунок ниже).



Таким образом, вызов геттер-метода `managedObjectContext` инициализирует весь стек объектов Core Data и приводит Core Data в готовность.

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.

2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Установить связь между интерфейсом приложения и логикой приложения, в соответствие с которой происходит обработка данных.
5. Собрать проект и запустить на симуляторе.

Вариант 1

1. Создать приложение «Библиотечный каталог», содержащие список книг.
2. Каждая книга должна иметь следующую информацию:

- Название.
- Автор.
- Регистрационный номер.
- Год издания.
- Жанр.
- Аннотация.
- Метка, информирующая о том, что данная книга была прочитана пользователем.

3. Реализовать работу приложения при помощи фреймворка Core Data.
4. Интерфейс приложения остается на усмотрение исполнителя данного задания.
5. Приложение должно содержать предзаполненную информацию о пяти книгах.
6. Реализовать возможность добавления книги и удаления книги из каталога.
7. Реализовать возможность отметки книги как прочитанной или непрочитанной, по умолчанию считать, что книга не является прочитанной.
8. Приложение должно выводить пользователю информацию в нескольких срезах:

- Список всех книг, с возможностью перехода к полному описанию книги.
- Список авторов, с возможностью перехода к списку книг каждого автора, и, в свою очередь, переход к полному описанию книги.
- Список, содержащий годы издания книг, с возможностью перехода к списку книг каждого года издания, и, в свою очередь, переход к полному описанию книг.
- Список прочитанных книг, с возможностью перехода к подробной информации о книге.
- Список непрочитанных книг, с возможностью перехода к подробной информации о книге.
- Список жанров, с возможностью перехода к списку книг каждого жанра и, в свою очередь, переход к подробной информации о книге.

9. Проверить работу приложения на симуляторе.

Вариант 2

1. Создать приложение «Фильмотека», содержащие список фильмов.
2. Каждый фильм должен иметь следующую информацию:

- Название.
- Режиссер.
- Уникальный код.
- Год выпуска.
- Жанр.
- Аннотация.
- Метка, информирующая о том, что данный фильм был просмотрен пользователем.
- Рейтинг фильма, установленный пользователем, в случае, если фильм был просмотрен им.

3. Реализовать работу приложения при помощи фреймворка Core Data.
4. Интерфейс приложения остается на усмотрение исполнителя данного задания.
5. Приложение должно содержать предзаполненную информацию о пяти фильмах.
6. Реализовать возможность добавления и удаления фильма из каталога.
7. Реализовать возможность отметки фильма как просмотренного или непросмотренного, по умолчанию считать, что фильм является просмотренным.
8. Если пользователь отметил, что данный фильм он смотрел, то дать возможность пользователю выставить рейтинг фильму по 10-бальной шкале (0 – не понравился, 10 – очень понравился).
9. Приложение должно выводить пользователю информацию в нескольких срезах:

- Список всех фильмов, с возможностью перехода к полному описанию фильма.
- Список режиссеров, с возможностью перехода к списку фильмов каждого режиссера, и, в свою очередь, переход к полному описанию фильма.
- Список, содержащий годы выпуска фильмов, с возможностью перехода к списку фильмов каждого года издания, и, в свою очередь, переход к полному описанию фильма.
- Список просмотренных фильмов, с сортировкой по рейтингу в порядке убывания (от 10 до 0) с возможностью перехода к подробной информации о фильме.
- Список непросмотренных фильмов, с возможностью перехода к подробной информации о фильме.
- Список жанров, с возможностью перехода к списку фильмов каждого жанра и, в свою очередь переход к подробной информации о фильме.

10. Проверить работу приложения на симуляторе.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Какие существуют способы сохранения данных в iOS приложении на устройстве?
2. Какие существуют директории, к которым имеет доступ разработчик при создании приложения?
3. Каковы основные компоненты фреймворка Core Data и для чего каждый компонент необходим?

Литература

1. Михаель Приват и Роберт Варнер, Pro Core Data for iOS, Apress, 2011.
2. Introduction to Core Data Programming Guide.
3. Дэйв Марк, Джек Наттинг, Джефф Ламарш. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch. Вильямс, 2012.

Лабораторная работа № 8
«Изучение класса NSURLConnection, работа с контейнерами данных XML и JSON»

Цель работы

1. Познакомиться с классом NSURLConnection.
2. Получить практические знания по работе с типами подключений к сети.
3. Научиться разбирать контейнеры данных XML и JSON.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Работа с сетью».

Теоретическая часть

Для проверки соединения с сетью необходимо использовать фреймворк SystemConfiguration, позволяющий определять состояние существующей сети из приложения, а также доступность указанного хоста. Фреймворк SCNetworkReachability реализует синхронную и асинхронную модели.

При работе с синхронной моделью, состояние сети можно узнать с помощью вызова функции `SCNetworkReachabilityGetFlags()`. Например:

```
static BOOL isNetworkReachable(void)  {
    BOOL returnValue = NO;

    //Инициализируется структура, которая будет описывать сокет для работы с протоколами IP
    struct sockaddr_in address;
    size_t address_len = sizeof(address);
    memset(&address, 0, address_len);
    address.sin_len = address_len;
    address.sin_family = AF_INET;

     //Определение переменной, которая будет отвечать за мониторинг сети
    SCNetworkReachabilityRef reachabilityRef = SCNetworkReachabilityCreateWithAddress(NULL, (const struct sockaddr*)&address);

    if (reachabilityRef != NULL) {
```

```
        SCNetworkReachabilityFlags flags = 0;
        if(SCNetworkReachabilityGetFlags(reachabilityRef, &flags)){
           //определение доступности сети
            BOOL isReachable = ((flags & kSCNetworkFlagsReachable) != 0);
            BOOL connectionRequired = ((flags & kSCNetworkFlagsConnec-
tionRequired) != 0);
            returnValue = (isReachable && !connectionRequired) ? YES :
NO;
        }
        CFRelease(reachabilityRef);
    }
    return returnValue;
}
```

Описание флагов приведено в документации, предоставленной компанией Apple.

При работе с асинхронной моделью необходимо реализовать обратную функцию, которая будет вызываться при изменении состояния сети, а затем зарегистрировать ее с помощью функции `SCNetworkReachabilitySetCallback()` и запланировать выполнение при помощи `SCNetworkReachabilityScheduleWithRunLoop()`, например:

```
typedef void (*SCNetworkReachabilityCallBack)     (
   SCNetworkReachabilityRef target,
   SCNetworkReachabilityFlags flags,
   void *info
);
```

Обратная функция:

```
static void _NetworkReachabilityCallback(SCNetworkReachabilityRef target,
SCNetworkReachabilityFlags flags, void* info) {
    @autoreleasepool {
        // Выполнение кода
    }
}
```

Реализация функций, запускающих и останавливающих мониторинг:

```
static SCNetworkReachabilityRef startMonitoring() {
    // Инициализируется структура, которая будет описывать сокет для ра-
боты с протоколами IP
    struct sockaddr_in address;
    size_t address_len = sizeof(address);
    memset(&address, 0, address_len);
    address.sin_len = address_len;
    address.sin_family = AF_INET;

    SCNetworkReachabilityRef reachabilityRef = SCNetworkReachabilityCre-
ateWithAddress(NULL, (const struct sockaddr*)&address);
```

```
    if (reachabilityRef != NULL) {
        SCNetworkReachabilityContext context = { 0, NULL, NULL, NULL,
NULL};
        if(SCNetworkReachabilitySetCallback(reachabilityRef,
_NetworkReachabilityCallback, &context)) {
            SCNetworkReachabilityScheduleWithRunLoop(reachabilityRef,
CFRunLoopGetCurrent(), kCFRunLoopDefaultMode);
        }
        return reachabilityRef;
    }
    return NULL;
}

static void stopMonitoring(SCNetworkReachabilityRef ref) {
    if(ref) {
        SCNetworkReachabilityUnscheduleFromRunLoop(ref, CFRunLoopGetCur-
rent(), kCFRunLoopDefaultMode);
        CFRelease(ref);
    }
}
```

Данное решение можно использовать в приложении следующим образом:

```
int main(int argc, char *argv[]) {
    SCNetworkReachabilityRef reachabilityRef = startMonitoring();
    if(reachabilityRef) {
        NSLog(Мониторинг запущен);
    } else {
        NSLog(Мониторинг не запущен);
    }

    //Код приложения

    stopMonitoring(reachabilityRef);
}
```

Компания Apple разработала класс, упрощающий всю процедуру проверки и мониторинга подключения – класс `Reachability`. Каждый созданный объект класса `Reachability` информирует о доступности конкретного вида подключения либо хоста.

Исходный код можно загрузить с сайта developer.apple.com, в разделе примеры.

Для проверки состояния соединения до конкретного хоста необходимо создать объект `Reachability` следующим образом:

```
Reachability* appleReachability = [Reachability reachabilityWithHostName:
@"www.apple.com"];
```

Для проверки наличия подключения к интернету необходимо создать `Reachablility` следующим образом:

```
Reachability* internetReachability = [Reachability reachabilityForInter-
netConnection];
```

Для проверки наличия подключения к Wi-Fi:

```
Reachability* wifiReachability = [Reachability reachabilityForLocalWiFi];
```

Для получения уведомления об изменении состояния подключения к сети нужно подписаться на `kReachabilityChangedNotification`:

```
[[NSNotificationCenter defaultCenter] addObserver: self selector:
@selector(reachabilityChanged:) name: kReachabilityChangedNotification
object: nil];
```

и определить метод, который будет «слушать» уведомление:

```
- (void) reachabilityChanged: (NSNotification* )note
{
    Reachability* curReach = [note object];
    NetworkStatus status = curReach.currentReachabilityStatus;

    switch (status) {
      case NotReachable:
        // хост недоступен
        break;
      case ReachableViaWiFi:
        // доступен через WiFi
        break;
      case ReachableViaWWAN:
        // доступен через 3G или EDGE
        break;
    }
}
```

После подписки на уведомления, нужно «попросить» объект `Reachability` генерировать уведомления об изменении состояния:

```
Reachability* internetReachability = <#объект Reachability#>;
[internetReachability startNotifier];
```

Для отправки запроса на сервер можно использовать класс `NSURLConnection`, позволяющий отправлять данные как синхронно, так и асинхронно, например:

```
NSString* attributes = @"param=value_1&other_param=58"; // задаются параметры POST запроса
NSURL* url = [NSURL URLWithString:@"http://myserver.com"]; // указывается ссылка куда необходимо произвести запрос
NSMutableURLRequest* request = [NSMutableURLRequest requestWithURL:url];
request.HTTPMethod = @"POST";
request.HTTPBody = [attributes dataUsingEncoding:NSUTF8StringEncoding];
// формируется строка запроса в необходимой кодировке
// отправляется запрос на сервер
[NSURLConnection sendSynchronousRequest:request returningResponse:nil error:nil];
```

Для передачи данных используются различные контейнеры данных, наиболее известными из них являются **XML и JSON**.

> ***XML*** *(eXtensible Markup Language, расширяемый язык разметки) – язык разметки, описывающий целый класс объектов данных, называемых XML-документами.*

Язык XML используется в качестве средства для описания грамматики других языков и контроля за правильностью составления документов, то есть сам по себе XML не содержит тэгов, предназначенных для разметки, он просто определяет порядок их создания.

Для разбора данных в формате XML на iOS рекомендуется использовать класс `NSXMLParser` на базе SAX. О нахождении каких-либо элементов парсер уведомляет свой делегат. Делегат должен следовать протоколу `NSXMLParserDelegate`. XML-парсер в Objective-C может работать как последовательно, так и асинхронно. Однако парсер предъявляет жесткие требования к формату данных и не принимает HTML-документы.

Методы, которые должен поддерживать делегат, следуя протоколу `NSXMLParserDelegate`:

```
//метод, вызывающийся в начале разбора документа
- (void) parserDidStartDocument:(NSXMLParser *)parser;

//метод, вызывающийся при полном прочитывании и разборе документа
- (void) parserDidEndDocument:(NSXMLParser *)parser;

//метод, вызывающийся в случае возникновения ошибки при разборе документа
- (void) parser:(NSXMLParser *)parser parseErrorOccurred:(NSError *)parseError;

//метод, вызывающийся в случае возникновения ошибки валидации данных
- (void) parser:(NSXMLParser *)parser validationErrorOccurred:(NSError *)validationError;

//метод, вызывающийся при нахождении нового элемента в документе
```

```
- (void) parser:(NSXMLParser *)parser
         didStartElement:(NSString *)elementName
         namespaceURI:(NSString *)namespaceURI
         qualifiedName:(NSString *)qName
         attributes:(NSDictionary *)attributeDict;

//метод, вызывающийся при нахождении закрывающего символа у элемента
- (void) parser:(NSXMLParser *)parser
         didEndElement:(NSString *)elementName
         namespaceURI:(NSString *)namespaceURI
         qualifiedName:(NSString *)qName;

//метод, считывающий по символам значение, хранящееся в элементе
- (void) parser:(NSXMLParser *)parser foundCharacters:(NSString )string;
```

В большинстве случаев создается свой класс-делегат для `NSXMLParser`, который собирает необходимую информацию из XML-документа, в случае возникновения ошибки делегат будет знать о ней. Так же делегат будет уведомлен об окончании разбора документа.

***JSON** (JavaScript Object Notation) – текстовый формат обмена данными, основанный на JavaScript.*

Несмотря на происхождение от подмножества языка стандарта ECMA-262 1999 года, формат считается язык независимым и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

До выхода iOS 5 встроенной поддержки разбора JSON не существовало. Однако SDK теперь включает класс `NSJSONSerialization`, поддерживающий преобразования документов JSON в объекты `Foundation` и наоборот. Документ JSON передается классу в виде объекта `NSData` с использованием статического метода класса `JSONObjectWithData:options:error:`:

```
NSError *error = nil;
id object = nil;
NSData *json = [NSData dataWithContentsOfFile:path];
object = [NSJSONSerialization JSONObjectWithData:json options:0 error:&error];
if( error ) {
 ... код обработки ошибок ...
}
```

Объект класса, возвращаемый методом класса `NSJSONSerialization`, представляет собой объект `NSArray` или `NSDictionary` в зависимости от структуры исходного документа

JSON. При создании объектов `Foundation` по данным JSON могут передаваться различные аргументы, в том числе:

- **аргумент `NSJSONReadingMutableContainers`** указывает, что массивы и словари создаются как изменяемые (mutable) объекты.
- **аргумент `NSJSONReadingMutableLeaves`** указывает, что листовые строки графа объектов JSON создаются как экземпляры `NSMutableString`.

Объект, который может быть преобразован в формат JSON, должен обладать **следующими свойствами:**

- Объект верхнего уровня должен относиться к типу `NSArray` или `NSDictionary`.
- Все объекты являются экземплярами классов `NSString`, `NSNumber`, `NSArray`, `NSDictionary` или `NULL`.
- Все ключи словаря являются экземплярами `NSString`.

Если объект удовлетворяет данным критериям, его можно преобразовать в JSON-формат следующим образом:

```
NSError *error = nil;
NSData *json = nil;
if( [NSJSONSerialization isValidJSONObject:object] ) {     //проверка: является ли переданный объект объектом для сериализации
 json = [NSJSONSerialization dataWithJSONObject:object options:0 error:&error];
} else {
 ... обработка недействительных объектов ...
}
if( error ) {
 ... код обработки ошибок ...
}
```

Помимо стандартных классов, представленных в iOS SDK, существует несколько сторонних библиотек, упрощающих настройку и подготовку проекта для работы с XML- и JSON-документами. Наиболее используемой на данный момент библиотекой является библиотека AFNetworking.

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.
2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Установить связь между интерфейсом приложения и логикой приложения, в соответствие с которой происходит обработка данных.
5. Собрать проект и запустить на симуляторе.

Вариант 1

1. Создать приложение с табличным представлением.
2. Данные должны загружаться с удаленного сервера при помощи XML-контейнера, можно использовать на выбор следующие ссылки (или создать собственную):

- http://www.w3schools.com/xml/cd_catalog.xml.
- http://www.w3schools.com/xml/plant_catalog.xml.
- http://www.w3schools.com/xml/simple.xml.

3. Данные необходимо отображать в полной мере в табличном представлении (верстка на выбор исполнителя).
4. Реализовать экран ожидания загрузки данных (т.е. в этот момент пользователь не может взаимодействовать с табличным представлением).
5. Загруженные данные необходимо сохранить на устройстве любым способом на выбор:

- с применением фреймворка CoreData;
- напрямую в базу данных SQLite;
- с использованием класса NSUserDefault;
- напрямую в файловую систему.

6. Реализовать проверку наличия доступа к сети и в случае отрицательного ответа, выводить сообщение, что необходимо подключение к сети.
7. Реализовать проверку на наличие данных, если данные присутствуют на устройстве, то загрузку данных производить не нужно.
8. Реализовать возможность удаления данных с устройства.
9. Реализовать ручную загрузку данных (т.е. при нажатии на кнопку, происходит проверка наличия доступа к сети и, в случае положительно ответа, по указанной ссылке загружаются данные, заменяя информацию, сохраненную на устройстве).

10. Проверить работу приложения на симуляторе.

Вариант 2

1. Создать приложение с табличным представлением.
2. Данные должны загружаться с удаленного сервера при помощи JSON-контейнера, можно использовать на выбор следующие ссылки (или создать собственную):

- http://api.geonames.org/citiesJSON?north=44.1&south=-9.9&east=-22.4&west=55.2&lang=de&username=demo.
- http://puppygifs.net/api/read/json.
- http://api.geonames.org/postalCodeSearchJSON?postalcode=40&maxRows=100&username=demo.

3. Данные необходимо отображать в полной мере в табличном представлении (верстка на выбор исполнителя).
4. Реализовать экран ожидания загрузки данных (т.е. в этот момент пользователь не может взаимодействовать с табличным представлением).
5. Загруженные данные необходимо сохранить на устройстве любым способом на выбор:

- с применением фреймворка CoreData;
- напрямую в базу данных SQLite;
- с использованием класса NSUserDefault;
- напрямую в файловую систему.

6. Реализовать проверку наличия доступа к сети и в случае отрицательного ответа, выводить сообщение, что необходимо подключение к сети.
7. Реализовать проверку на наличие данных, если данные присутствуют на устройстве, то загрузку данных производить не нужно.
8. Реализовать возможность удаления данных с устройства.
9. Реализовать ручную загрузку данных (т.е. при нажатии на кнопку, происходит проверка наличия доступа к сети и в случае положительно ответа по указанной ссылке загружаются данные, заменяя информацию, сохраненную на устройстве).
10. Проверить работу приложения на симуляторе.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Какие типы подключений к сети можно реализовать в iOS приложении на устройстве?
2. Какой механизм разбора XML-документов реализован в iOS SDK?
3. Какими свойствами должен обладать объект, чтобы его можно было преобразовать в JSON формат?

Литература

1. Дэйв Марк, Джек Наттинг, Джефф Ламарш. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch. Вильямс, 2012.
2. Документация Apple.
3. Михаель Приват и Роберт Варнер, Pro Core Data for iOS, Apress, 2011.

Лабораторная работа № 9

«Изучение сервиса геолокации»

Цель работы

1. Познакомиться с классами геолокации данных в iOS.
2. Получить практические знания по определению местоположения пользователя в приложениях для iOS-устройств.
3. Научиться строить маршрут в приложении.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Сервис геолокации».

Теоретическая часть

Любое iOS-устройство обладает возможностью ориентироваться в окружающем пространстве, используя для этой цели библиотеку Core Location. Данная библиотека взаимодействует с четырьмя аппаратными модулями, от которых получает необходимую информацию (см. рисунок ниже).

GPS

- глобальная система навигации и определения местоположения, обеспечивающая измерение расстояния, времени и определения местоположения во всемирной системе координат WGS 84.

ГЛОНАСС

- глобальная навигационная спутниковая система, разработанная по заказу Министерства обороны СССР, предназначенная для оперативного навигационно-временного обеспечения числа пользователей наземного, морского, воздушного и космического базирования.

Триангуляция ретрансляторов сотовой связи

- определение местоположение пользователя по трем базовым станциям сотовых операторов путем рассчета расстояния до мобильного телефона, исходя из местоположения вышек ретрансляторов в зоне действия сотовой связи.

Wi-Fi Positioning Sistem (WPS)

- навигационная система позиционирования, основывающаяся на определении координат по Wi-Fi точкам.

Рисунок 37. Список аппаратных модулей для геопозиционирования устройства в пространстве

Самыми точными из вышеперечисленных модулей являются модули GPS и ГЛОНАСС. Триангуляция ретрансляторов может быть достаточно точной в городских условиях или в районах с большой плотностью расстановки базовых станций, но ухудшается в тех районах, где вышки ретрансляторы расставлены реже. Система WPS использует MAC-адреса ближайших точек доступа беспроводной связи для приблизительного определения местоположения путем обращения к базам данных известных поставщиков услуг беспроводной связи и зоны их обслуживания.

Необходимо иметь в виду, что при работе с библиотекой Core Location повышается уровень расхода заряда батареи мобильного устройства, в связи с чем необходимо контролировать работу определения местоположения и не запускать процессы при отсутствии необходимости в этом. При работе с Core Location можно задавать желательную точность определения местоположения, чем появляется возможность существенно сэкономить энергию заряда батареи.

При работе с сервисом геолокации наиболее часто разработчики имеют дело с классом `CLLocationManager`, который является диспетчером местоположения. Для работы с ним необходимо создать экземпляр класса:

```
CLLocationManager *locationManager = [CLLocationManager new];
```

Для начала опроса местоположения пользователя данным экземпляром класса необходимо создать объект, который будет соответствовать протоколу `CLLocationManager`, и назначить его в качестве делегата диспетчера местоположения.

```
locationManager.delegate = self;
```

Для определения точности определения местоположения (в некоторых случаях достаточно определить регион или страну), необходимо использовать тип `CLLocationAccuracy`, который определяется как тип данных `double`.

```
//с точностью до 1 километра
locationManager.desiredAccuracy = kCLLocationAccuracyKilometer;
```

По умолчанию диспетчер местоположения уведомляет делегат о любых обнаруженных изменениях в местоположении устройства. Для экономии ресурсов необходимо использовать фильтр расстояния, который укажет делегату, в каких случаях необходимо производить уведомления. Фильтры расстояния задаются в метрах.

```
// уведомлять приложение только в том случае, если устройство сместилось
более чем на 12 метров
locationManager.distanceFilter = 12;
```

Можно указать дополнительные настройки для диспетчера местоположения.

Например: *автоматическую остановку/возобновление сервиса определения местоположения или для различных положений устройства указать, какая из границ будет являться ориентиром на север.*

После того как все настройки будут произведены, останется только запустить работу диспетчера местоположения при помощи следующего метода:

```
[locationManager startUpdatingLocation];
```

После запуска диспетчер местоположения перейдет в фоновый режим работы и при определении нового местоположения вызовет необходимый метод делегата. Работа диспетчера местоположения будет продолжаться до тех пор, пока не будет послано сообщение об остановке:

```
[locationManager stopUpdatingLocation];
```

Сведение о местоположении передаются от диспетчера местоположения экземпляру класса `CLLocation`:

```
CLLocation *location;
```

который содержит следующую информацию:

- координаты местоположения (широта и долгота):

```
CLLocationDegrees latitude = location.coordinate.latitude;
CLLocationDegrees longitude = location.coordinate.longitude;
```

- уровень точности определения делегатом местоположения широты и долготы:

```
CGFloat horizontalAccuracy = location.horizontalAccuracy;
CGFloat verticalAccuracy = location.verticalAccuracy;
```

- высоту местности над уровнем моря:

```
CLLocationDistance = location.altitude;
```

Если система `Core Location` не может определить текущее местоположение устройства, то вызывается метод делегата `locationManager:didFailWithError:`.

Более подробную информацию о работе сервиса геолокации можно получить в официальной документации Apple – Location and Maps Programming Guide.

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.
2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Установить связь между интерфейсом приложения и логикой приложения, в соответствии с которой происходит обработка данных.

5. Собрать проект и запустить на симуляторе.

Задача

1. Создать приложение с представлением карты (MKMapView).
2. Добавить элементы управления на данном экране, которые будут выполнять следующие действия:

- Переключать отображение карты в режим «спутника».
- Переключать отображение карты в режим «стандарт».
- Переключать отображение карты в режим «гибрид».
- Возможность включить режим «компаса» (карта ориентируется не на север, а на направление пользователя).

3. Приложение должно уметь определять местоположение пользователя (если при выполнении задания используется симулятор, то указать координаты своего местоположения).
4. В отдельном представлении выводить информацию (после определения местоположения пользователя):

- Координаты пользователя (широта, долгота).
- Город, в котором находится пользователь.
- Улица, на которой находится пользователь.
- Ближайший номер дома, рядом с пользователем.

5. Реализовать возможность установить на карте в любом месте «булавку» с текстовой пометкой, и до этой точки необходимо проложить маршрут от местоположения пользователя.
6. Если по каким-то причинам невозможно выполнить вышеуказанные действия, необходимо выводить уведомление с указанием причины.
7. Проверить работу приложения на симуляторе.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. Какой класс из iOS SDK необходимо использовать, чтобы по координатам пользователя определить адрес?

2. Какими способами в iOS возможно определение местоположения устройства?

Литература

1. Дэйв Марк, Джек Наттинг, Джефф Ламарш. iOS 5 SDK. Разработка приложений для iPhone, iPad и iPod touch. Вильямс, 2012.
2. Документация Apple.

Лабораторная работа № 10
«Модульное тестирование»

Цель работы

1. Познакомиться с модульным тестированием.
2. Научиться создавать и применять модульные тесты в iOS-приложениях.

Программное обеспечение

1. IDE XCode.
2. iOS SDK 7.

Необходимая теоретическая подготовка

- Освоить лекционный материал по теме: «Юнит-тестирование».

Теоретическая часть

Модульное тестирование (unit testing) *заключается в изолированной проверке каждого отдельного элемента путем запуска тестов в искусственной среде.*

Поэлементное тестирование позволяет оценивать каждый элемент изолированно и, подтверждая корректность его работы, точечно установить проблему значительно проще, чем если бы тестируемый элемент был частью системы.

При реализации модульных тестов используются **заглушки**, которые заменяют недостающие компоненты в вызываемом элементе и выполняют **следующие действия:**

- возвращаются к элементу, не выполняя никаких других действий;
- отображают трассировочное сообщение и иногда предлагают тестеру продолжить тестирование;
- возвращают постоянное значение или предлагают тестеру самому ввести возвращаемое значение;
- осуществляют упрощенную реализацию недостающей компоненты;
- имитируют исключительные или аварийные условия.

Модульное тестирование является одной из ключевых практик методологии экстремального программирования.

Преимущества модульного тестирования:

- написание тестов помогает войти в рабочий ритм;
- придает уверенность в работоспособности кода;
- дает запас прочности при дальнейшей интеграции или изменениях кода.

Ключевым фактором при оценке перспективности любого метода является стоимость проекта. Дополнительная работа по созданию тестов, их кодированию и проверке результатов вносит существенный вклад в общую стоимость проекта. Бездумное применение тотального модульного тестирования почти гарантированно приведет к получению неоптимального продукта.

Модульное тестирование оправдано, если в итоге будет получен следующий результат:

- снизится время на отладку приложения;
- появится возможность поиска ошибок с меньшими затратами по сравнению с другими подходами;
- появится возможность дешевого поиска ошибок при изменениях кода в дальнейшем.

Если в результате исправления ошибок интеграции меняется исходный код, в нем с большой вероятностью появляются ошибки. Если в результате добавления новой функциональности меняется исходный код, в нем с большой вероятностью появляются ошибки. Выполнять поиск возникших ошибок лучше и удобнее с помощью ранее созданных модульных тестов.

Цель модульного тестирования: получение работоспособного кода с наименьшими затратами. Его применение оправдано тогда и только тогда, когда оно более эффективно, чем другие методы.

Таким образом, при создании модульных тестов для приложения необходимо руководствоваться следующими принципами:

- не имеет смысла писать тесты на весь исходный код, так как некоторые ошибки проще найти на более поздних стадиях разработки программы. Эффективней написать тесты на вызывающий класс и создать тесты, проверяющие все участки кода;
- писать тесты необходимо для исходного кода, который потенциально подвержен изменениям;
- для избежания частого изменения тестов следует ответственно подойти к процессу планирования интерфейсов и, соответственно, к написанию исходного кода.

Разработка через тестирование требует от разработчика создания автоматизированных модульных тестов, определяющих требования к коду непосредственно перед написанием самого кода. Тест содержит проверки условий, которые могут либо выполняться, ли-

бо нет. Прохождение теста подтверждает поведение, предполагаемое программистом. Разработчики часто пользуются различными библиотеками (frameworks) для создания и автоматизации запуска наборов тестов. К тестам применяются те же требования стандартов кодирования, что и к основному коду.

Цикл разработки с использованием модульных тестов представлен на рисунке ниже. Подробнее о каждом этапе написано далее.

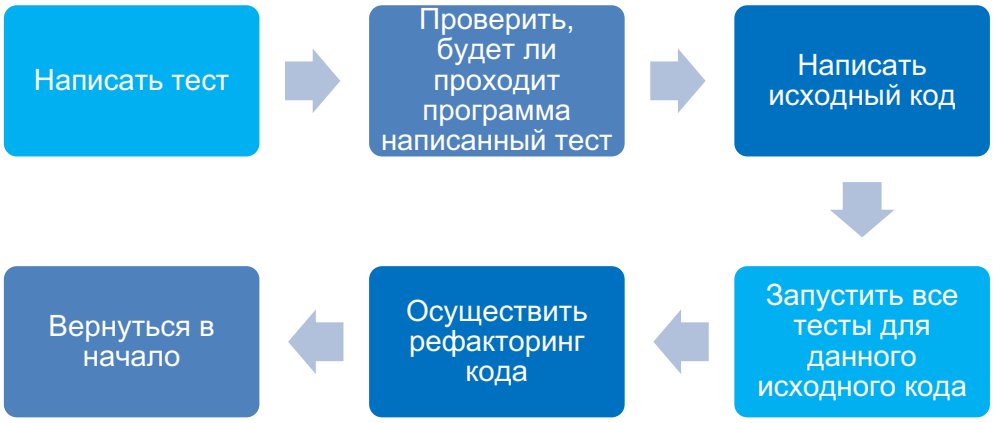


Рисунок 38. Цикл разработки в случае применения модульных тестов

1. **Добавление теста.** Добавление каждой новой функции в приложение начинается с написания теста. Чтобы написать тест, разработчик должен четко понимать предъявляемые к новой функции требования, для чего рассматриваются возможные сценарии использования и пользовательские истории. Новые требования могут также повлечь изменение существующих тестов.
2. **Проверка написанных тестов.** На этом этапе проверяется, что только что написанные тесты программа не проходит, поскольку соответствующий исходный код еще не написан (если же созданный тест прошел проверку, это означает, что-либо предложенная «новая» функция уже существует, либо тест имеет недостатки). Также проверяются сами тесты: написанный тест может проходить всегда и соответственно быть бесполезным. Этот этап предназначен для того, чтобы определить, что тест действительно тестирует то, для чего он был разработан.
3. **Написание исходного кода**. На этом этапе новый код пишется с тем условием, что он пройдет тест. Этот код не обязательно должен быть идеален, допустимо, чтобы он проходил тест каким-то неэлегантным способом – на последующих этапах код будет улучаться. Важно писать код, предназначенный именно для прохождения теста, – не следует добавлять лишней и, соответственно, не тестируемой функциональности.
4. **Запуск всех тестов**. Необходимо убедиться, что написанный код проходит все тесты. Только в этом случае, разработчики могут быть уверены, что код удовлетворяет всем

тестируемым требованиям. После этого можно приступить к заключительному этапу цикла.

5. **Рефакторинг**. Улучшение кода после достижения требуемой функциональности.
6. **Повторить цикл**. Описанный цикл повторяется, реализуя все новую и новую функциональность. Шаги следует делать небольшими, от 1 до 10 изменений между запусками тестов. Если новый код не удовлетворяет новым тестам или старые тесты перестают проходить, программисту необходимо вернуться к отладке приложения. При использовании сторонних библиотек не следует делать настолько небольшие изменения, которые буквально тестируют саму стороннюю библиотеку, а не код, ее использующий, если только нет подозрений, что библиотека содержит ошибки.

Набор тестов должен иметь доступ к тестируемому коду, но не следует забывать, что принципы инкапсуляции и сокрытия данных нарушаться не должны. Поэтому модульные тесты обычно пишутся в том же модуле или проекте, что и тестируемый код.

Важно, чтобы фрагменты кода, предназначенные исключительно для тестирования, не оставались в выпущенном приложении. Для этого необходимо использовать директивы условной компиляции. Однако это будет означать, что выпускаемый код не будет полностью совпадать с протестированным. Систематический запуск интеграционных тестов на выпускаемой сборке поможет удостовериться в том, что не осталось кода, скрыто полагающегося на различные аспекты модульных тестов.

При создании проекта в XCode 5.x шаблон файла для модульных тестов автоматически добавляется в проект. Самостоятельно добавить тест-файлы можно через меню *File → New → File → Cocoa Touch → Objective-C test case class*. Модульные тесты запускаются в отдельном меню targets (цели) проекта, который обычно называется «*НазваниеПроекта*Tests». К данной цели (Targets) дополнительно необходимо добавить файл с исходным кодом тестируемого класса.

Исходный код файла тестов выглядит следующим образом:

```
//ExtendedString_Test.m

#import <XCTest/XCTest.h> //используемый тестовый фреймворк,
//в котором необходимо указать класс, который будет тестироваться

@interface NameProjectTests : XCTestCase //Все тест-классы должны наследоваться от данного класса
//описание переменных класса, свойств и дополнительных методов
@end

@implementation NameProjectTest
```

```
- (void)setUp  //данный метод запускается первым, в нем можно настроить тестируемые классы и создать дополнительные объекты
{
    [super setUp];
}

- (void)tearDown  //данный метод запускается последним, в нем можно удалить объекты
{
    [super tearDown];
}

- (void)testExample  //обычный тестовый метод, в котором можно описать сам тест
{
    XCTFail (@"No implementation for \"%s\"", __PRETTY_FUNCTION__);
}

@end
```

Во время сборки targets с тестами создается специальный бандл с расширением *.xctest. Следующим шагом в сборке этого targets является запуск скрипта следующего содержания:

```
${SYSTEM_DEVELOPER_DIR}/Tools/RunUnitTests
```

Приведенный выше скрипт в свою очередь запускает вложенные скрипты, но все, в конечном итоге, сводится к вызову `/Developer/Tools/xtest` с определенными параметрами. Последний загружает в себя созданный тестовый бандл, находит там классы-наследники `XCTestCase` и начинает выполнять методы в начале `setUp`, в конце `tearDown`, а между ними он вызывает все методы, название которых начинается со слова «`test`». Поэтому все тесты нужно начинать с приставки «`test`».

Если во время выполнения тестов не будет обнаружено ошибок, то сборка targets Tests пройдет успешно. Если какой-то из тестов окажется не пройденным, то во время сборки будет выведена ошибка (подобная ошибке компиляции).

Более подробную информацию о применении модульных тестов в iOS-приложениях можно получить в официальной документации, представленной компанией Apple – Unit Test Your App.

Практическая часть

Постановка задачи

Создать в XCode проект iOS-приложения, выполняющего указанные действия.

Порядок выполнения работы

1. Проанализировать задание, создать проект на основе любого шаблона, добавить необходимые классы Objective-C.
2. В заголовочном файле (*.h) класса объявить необходимые переменные, если нужно, объявить также свойства и прототипы методов.
3. В файле реализации (*.m) класса описать работу методов класса и экземпляров класса.
4. Установить связь между интерфейсом приложения и логикой приложения, в соответствии с которой происходит обработка данных.
5. Собрать проект и запустить на симуляторе.

Вариант №1

1. Создать приложение на основе любого шаблона с поддержкой unit-тестов.
2. Продумать и создать тест, проверяющий результат умножения и сложения двух чисел.
3. Проверить написанный тест, показать, как он работает в случае неправильных данных, и показать его работу в случае корректно введенных данных.

Вариант №2

1. Создать приложение на основе любого шаблона с поддержкой unit-тестов.
2. Продумать и создать тест проверяющий результат инверсии строки (когда строка написана наоборот – первый символ является последним, следующий символ написан предпоследним и т.д.).
3. Проверить написанный тест, показать, как он работает в случае неправильных данных, и показать его работу в случае корректно введенных данных.

Отчет о выполнении работы

1. Подробное описание выполненной работы со скриншотами предоставить в текстовом документе.
2. Результаты работы вложить в архив (в формате *.zip).

Вопросы для самопроверки

1. В чем заключается цель модульного тестирования?
2. Какие этапы содержит цикл разработки с использованием модульных тестов?
3. Какие методы применяются при создании тестов и каково их назначение?

Литература

1. Грэхем Ли. Разработка через тестирование для iOS. ДМК Пресс, 2012, 272 с..
2. Документация Apple.