12 Объекты

Во многих языках программирования поддерживается концепция *объектов*. Объекты, как и структуры, содержат некоторые данные. Однако в отличие от структуры, объект также содержит набор функций для работы с этими данными. Чтобы вызвать такую функцию, следует отправить объекту *сообщение*. В объектной терминологии функция, вызываемая при помощи сообщения, называется *методом*.

В начале 1980-х годов Брэд Кокс и Том Лав решили включить объектно-ориентированные возможности в язык C. Они использовали идею выделения памяти в куче и дополнили ее синтаксисом отправки сообщений. Так появился язык Objective-C.

Объекты по своей природе очень «общительны». В ходе своей работы они постоянно принимают и отправляют сообщения, касающиеся всего, что они делают. Сложная программа на языке Objective-C может одновременно хранить в памяти сотни объектов, которые работают и отправляют сообщения друг другу.

Класс описывает конкретную разновидность объектов. В описание включаются методы и *переменные экземпляров*, в которых объекты некоторого типа хранят свои данные. Программист обращается к классу с запросом о создании объекта его типа в куче. Полученный объект называется *экземпляром* этого класса.

Например, на iPhone установлено множество классов, одним из которых является класс `CLLocation`. Вы можете обратиться к классу `CLLocation` с запросом на создание экземпляра `CLLocation`. Объект `CLLocation` содержит несколько переменных для хранения географических данных: широты, долготы и высоты. Объект также содержит набор методов. Например, можно «спросить» у одного экземпляра `CLLocation`, на каком расстоянии он находится от другого объекта `CLLocation`.

Создание и использование объектов

Пришло время написать нашу первую программу на Objective-C. Создайте новый проект командной строки (`Command Line Tool`), но вместо типа `C` выберите тип `Foundation` (рис. 12.1). Присвойте проекту имя `TimeAfterTime`.

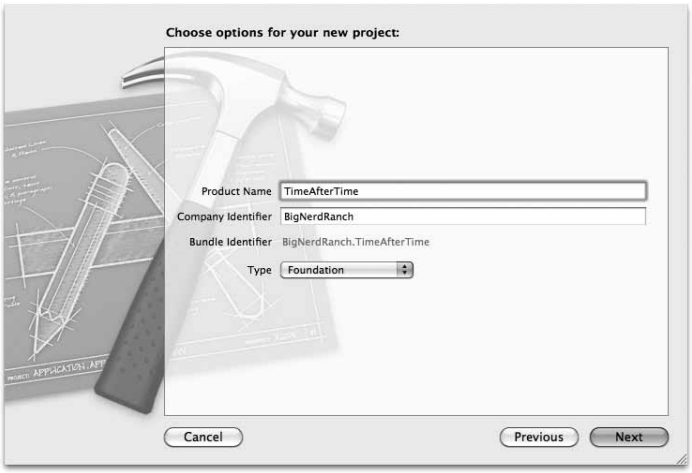

Рис. 12.1. Создание программы командной строки Foundation

Файлы с кодом Objective-C обычно имеют суффикс *.m*. Найдите и откройте файл *main.m*, введите следующие две строки кода:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSDate *now = [NSDate date];
        NSLog(@"The new date lives at %p", now);

    }
    return 0;
}
```

Поздравляю — ваше первое сообщение успешно отправлено! Вы отправили сообщение `date` классу `NSDate`. Метод `date` приказывает классу `NSDate` создать экземпляр `NSDate`, инициализировать его текущей датой/временем и вернуть адрес, с которого размещается в памяти новый объект. Полученный адрес

охраняется в переменной now. Соответственно переменная является указателем на объект NSDate.

NSLog() — функция Objective-C, отчасти напоминающая printf(); она тоже получает форматную строку, заменяет заполнители % фактическими значениями и выводит результат на консоль. Однако ее форматная строка всегда начинается с символа @ и ее не нужно завершать символом новой строки \n.

Постройте и запустите программу. Результат должен выглядеть примерно так:

```
2011-08-05 11:53:54.366 TimeAfterTime[4862:707] The new date lives at
0x100114dc0
```

В отличие от printf() функция NSLog() выводит перед своими данными дату, время, имя программы и идентификатор процесса.

В дальнейшем, приводя результаты вызова NSLog(), я буду опускать эти служебные данные — строки получаются слишком длинными.

В NSLog() заполнитель %p выводит адрес объекта. Чтобы результат больше напоминал дату, используйте заполнитель %@, который приказывает объекту вывести свое описание в виде строки:

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {
        NSDate *now = [NSDate date];
        NSLog(@"The date is %@", now);

    }
    return 0;
}
```

Новый результат будет выглядеть примерно так:

```
The date is 2011-08-05 16:09:14 +0000
```

Анатомия сообщений

Отправка сообщения всегда заключается в квадратные скобки и всегда состоит из минимум двух частей:

- указатель на объект, получающий сообщение;
- имя вызываемого метода.

Отправка сообщения (как и вызов функции) может иметь аргументы. Рассмотрим пример.

Объекты NSDate представляют конкретную дату и время. Экземпляр NSDate может сообщить разность (в секундах) между датой/временем, которые представляет он сам, и временем 0:00 (по Гринвичу) 1 января 1970 года. Чтобы получить эту информацию, следует отправить сообщение timeIntervalSince1970 объекту NSDate, на который ссылается указатель now.

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSDate *now = [NSDate date];
        NSLog(@"The date is %@", now);
        double seconds = [now timeIntervalSince1970];
        NSLog(@"It has been %f seconds since the start of 1970.",
                seconds);

    }
    return 0;
}
```

Допустим, вам понадобился новый объект date — соответствующий моменту времени на 100 000 секунд позднее уже существующего. Отправив это сообщение исходному объекту date, вы получите новый объект date. Метод получает аргумент: величину смещения в секундах. Воспользуемся им для создания нового объекта date в функции main():

```
#import <Foundation/Foundation.h>

int main (int argc, const char * argv[])
{
    @autoreleasepool {

        NSDate *now = [NSDate date];
        NSLog(@"The date is %@", now);

        double seconds = [now timeIntervalSince1970];
        NSLog(@"It has been %f seconds since the start of 1970.",
                seconds);

        NSDate *later = [now dateByAddingTimeInterval:100000];
        NSLog(@"In 100,000 seconds it will be %@", later);

    }
    return 0;
}
```

В отправляемом сообщении `[now dateByAddingTimeInterval:100000]`:

- `now` — указатель на объект, получающий сообщение («получатель»);
- `dateByAddingTimeInterval` — имя метода («селектор»);
- 100 000 — единственный аргумент.

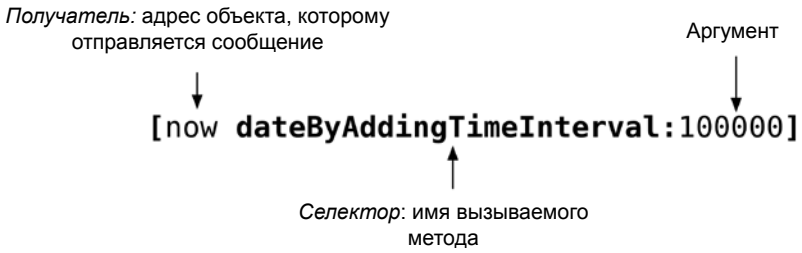

Рис. 12.2. Отправка сообщения

Объекты в памяти

На рис. 12.3 изображена диаграмма объектов. На ней изображены два экземпляра `NSDate` в куче. Две переменные `now` и `later` являются частью кадра функции `main()`. Они указывают на объекты `NSDate` (связи изображены стрелками).

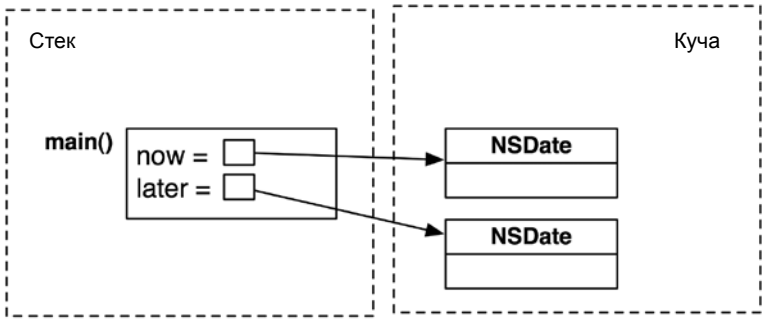

Рис. 12.3. Диаграмма объектов приложения TimeAfterTime

Пока что мы видели только один класс: `NSDate`, но в iOS и Mac OS X входят сотни классов. С самыми распространенными из них мы познакомимся в следующих главах.

id

При объявлении указателя на объект обычно указывается класс объекта, на который будет ссылаться этот указатель:

```
NSDate *expiration;
```

Но время от времени бывает нужно создать указатель, не зная точно, на какой объект он будет ссылаться. В таких случаях используется тип `id`, который означает «указатель на какой-либо объект Objective-C». Пример использования `id`:

```
id delegate;
```

Обратите внимание на отсутствие звездочки (*) в этом объявлении — она подразумевается при наличии `id`.

На первых порах довольно трудно усвоить концепции классов, объектов, сообщений и методов. Не огорчайтесь, если вы немного неуверенно чувствуете себя при работе с объектами — ведь это только начало. Мы будем использовать эти конструкции снова и снова, и с каждым разом они будут выглядеть все более осмысленно.

Упражнение

Используйте два экземпляра `NSDate` для вычисления продолжительности вашей жизни в секундах. Подсказка: новый объект `date` по заданному году, месяцу и т. д. создается следующим образом:

```
NSDateComponents *comps = [[NSDateComponents alloc] init];
[comps setYear:1969];
[comps setMonth:4];
[comps setDay:30];
[comps setHour:13];
[comps setMinute:10];
[comps setSecond:0];
NSCalendar *g = [[NSCalendar alloc]
                  initWithCalendarIdentifier:NSGregorianCalendar];
NSDate *dateOfBirth = [g dateFromComponents:comps];
```

Для получения количества секунд между двумя экземплярами `NSDate` следует использовать метод `timeIntervalSinceDate`:

```
double d = [laterDate timeIntervalSinceDate:earlierDate];
```