Objective-C

Why Learn Objective-C?

Objective-C 는 코코아를 구현한 언어입니다. 이 때문에 Objective-C를 이해하는 것은 코코아를 이해하는데 매우 중요합니다. 더불어 언어 이해가 코코아의 디자인 철학을 이해하는데 도움이 될 것입니다. 다수의 코코아 예제들 역시 Objective-C로 구현되어 있습니다.

Objective-C는 ANSI C 프로그래밍의 상위 개념에 속합니다. 그 결과 Objective-C 코드는 C 코드와 쉽게 섞입니다. C 언어로 작성된 OS의 기능들을 사용해야 하는 객체지향 Application 을 구현할 수 있도록 해줍니다. Objective-C로 만들어진 Application도 Mac OS X OS의 모든 기능을 사용할 수 있습니다.

Additions to C

Objective-C 가 C 언어의 확장판이므로 C 로 작성하는 프로그래밍 할때 사용했던 모든 지식들을 활용할 수 있습니다. Objective-C 를 사용할때 쓰게 되는 C 언어의 부분을 작지만 언제든지 사용할 수 있습니다. Objective-C에는 몇가지의 새로운 타입, 키워드, 관용구를 추가했습니다. 새로운 추가 사항들은 간단한 디자인이지만 매우 강력합니다. 또한 C 언어와 OOP의 개념을 알고 있다면 간단하게 몇분 안에 핵심사항을 익힐 수 있습니다.

Objective-C의 핵심은 객체간에 메시지를 주고 받는 개념입니다. Objective-C는 메시지를 언어의 한 요소로서 추가하였습니다.

Messaging

메시지 전송이 Objective-C를 동적으로 만들어줍니다. 동적이라는 개념에 대해서 짐작하지 마십시요. 차츰 차츰 알게 될겁니다.

메시지는 오브젝트에게 뭔가 하도록 만듭니다. Objective-C에서 메시지를 보내는 방식은 다음과 같습니다.

```
[someObject doSomething]
```

사각꺽쇠([,]) 메시지구문의 시작과 끝을 가리킵니다. 변수인 someObject는 메시지 수신자입니다. 변수 doSomething 는 Selector라고 부르고 메시지를 보냅니다. 메시지 전송은 항상 아래와 같은 형태를 갖습니다.

```
[receiver selector]
```

메시지는 임의의 argument나 리턴값을 가질 수 있습니다. 또한 메시지는 어느 수신자에게로도 전송이 가능합니다. 수신자가 메시지를 알지 못하는 경우에는 런타임에러가 발생됩니다.

어쨋든, 에러를 피하는건 쉽습니다. 실행시간에 특정 메시지를 보내기 전에 어떤 리시버가 메시지를 수신할 수 있는지 결정 할 수 있기 때문입니다.

메시지 구문은 처음에는 낯설겁니다. Objective-C의 메시지 전송 구문이 함수 호출 구문에 비해 독특한 것은 함수 호출구문과 분명히 다른 처리이기 때문입니다. 이 두종류의 개념을 같은 방식으로 표현한다면 구문상 오해를 만들 수 있습니다.

메시지 전송이 유연하면서 동적인 이유는 receiver와 selector 가 가변이기 때문입니다. 메시지가 어느 수신자에게 가게 될지 결정하는 것은 프로그램이 실행되기 전까지 결정되지 않습니다. 프로그램이 컴파일되는 시점에는 어느 오브젝트나 오브젝트의 타입을 모를 수도 있습니다. 컴파일 시점에 selector 도 알수 없습니다. selector 는 동적으로 로드된 오브젝트나 유저의 입력에 의해서도 추가될 수 있습니다. 메시징방식 덕분에 Objective-C 언어는 스크립트언어와 같은 동적인 언어와 비교적 쉽게 결합됩니다.

메시지 전송 방식은 상당히 동적이라서 receiver는 메시지를 전송한 코드를 가진 application이 아닐 수도 있습니다. 같은 컴퓨터 또는 다른 컴퓨터간에도 전송될 수 있습니다. 이경우 분산메시지라 불리는 것을 사용합니다. 분산메시지의 구문은 local에서 사용하는 메시지 구문과 문법이 동일합니다. 사실 컴파일러는 분산메시지와 일반메시지를 컴파일하는 동안 알 수 없습니다.

메시지 전송의 실제 구현은 단순하고 작고 효과적으로 이루어집니다. 대부분 언어가 프로그램 스택과 힙을 런타임에 초기화하는 main()과 프로그램 진입점(EntryPoint)를 갖고 있습니다. (번역이상) Objective-C에서의 런타임은 좀 더 주요한 역할이 있습니다. Objective-C 런타임은 프로그램의 실행 내내 좀 더 active 하며 초기화보다는 많은 일을 합니다. (/번역이상)

File Naming and Importing

Objective-C 파일은 .m 확장자를 사용해서 저장합니다. 이 확장자를 사용하면 컴파일러는 Objective-C 코드라는 것을 알게됩니다. (gcc의 경우에) Objective-C Application 은 .h 확장자를 갖는 헤더파일과 .m 확장자를 갖는 소스코드로 작성됩니다.

Objective-C 에서 다른 헤더를 포함시키려면 C 언어에서 #include 전처리문을 사용하듯이 #import를 사용합니다. #import 는 #include와 유사합니다만 파일이 중복해서 포함되지 않도록 처리해줍니다. 따라서 중복 포함을 방지하는 보완장치 (보통 #define, #ifndef, #endif) 를 넣을 필요가 없습니다. 코코아 헤더를 사용하는 경우에는 꼭 #import를 사용해야 합니다. 코코아 헤더들은 중복 포함 방지 보완코드가 없습니다.

Note

#import 에 더해서, Objective-C 에서도 // 식의 주석이 허용됩니다. C++과 C의 변종들과 마찬가지로 // 부터 줄끝까지 주석으로 간주합니다.

The id Type

Objective-C 가 OO 를 지원하는 언어로서 메시지를 받는 모든 것을 객체로 정의합니다. 메시지를 받는 receiver 는 컴파일 시점에 정의되지 않아도 됩니다. 메시지를 보내는 시점에 필요한 것은 receiver가 객체면 됩니다.

Objective-C 는 객체의 포인터를 id 라는 새로운 타입으로 정의합니다. id 로 선언된 변수는 어떤 메시지도 받을 수 있습니다. id 타입은 C 언어의 (void*)와 비슷하며 실제로 컴파일러가 메모리의 일부분만 참조한다는 점도 같습니다.

id 타입 다른 C 언어의 타입과 동일하게 사용됩니다. 아래 코드는 id 타입의 변수를 선언합니다.

```
id anObject;
```

변수로 선언된 anObject는 어떤 오브젝트의 포인터가 됩니다. C 에서 어떤 포인터든 상수값인 NULL 로 설정할 수 있습니다. Objective-C 에선 상수값 nil 로 설정할 수 있습니다. nil 로 메시지를 보내는 것은 에러가 아닙니다. 이 경우 주의 사항은 nil 로 보낸 메시지의 리턴값이 경우에 따라 무정의 상태가 될 수 있습니다. (오동작할 수 있다는 이야기라면 더 심각한 것 아닌가?)

static typing

id 타입은 객체에 대해 잘 모르거나 언어 레벨의 유연성이 필요할때만 사용되어야 합니다. (왠지 OverHead가 있을 것 같다) 컴파일러에게는 가능한 객체에 대해서 자세히 알려줘야 합니다. 객체에 대해 알고 있는 경우에는 정적인 변수를 선언해서 해당 항목에 대해 컴파일러에게 알려줘야 합니다.

정적인 변수 선언을 하려면 간단히, 특정 클래스의 인스턴스의 포인터로 선언하면 됩니다. 예를 들어 NSString 라는 클래스가 있다면 , NSString 의 인스턴스를 저장할 포인터는 다음과 같이 선언됩니다 :

```
NSString *theString;
```

메시지 수신자로서 theString 를 만날때마다 컴파일러는 theString이 메시지를 처리할 수 있는지 타입정보를 사용할 수 있습니다. 전송된 메시지를 처리할 수 있는 어떤 것도 발견하지 못한 경우 '경고'가 발생합니다. <이상함>컴파일러가 에러 대신 경고를 만드는 것은 메시지

자체를 이해하지 못했기 때문입니다.</이상함> 이건 수신측에서는 메시지를 이해했지만 컴파일러가 검증 할 수 있는 방법이 아직 없는 경우가 있을 수 있기 때문입니다. (내가 이해가 안된다)

아래의 타입들은 Objective-C runtime 에 정의된 타입들이며 프로그램 작성시 사용가능합니다.

- SEL : selector 를 저장하는데 사용됨
- IMP : 메시지를 처리하는 함수의 포인터를 저장하는데 사용함
- Class : Objective-C 의 클래스 오브젝트 포인터를 저장하는데 사용함
- id : 임의로 생성된 Objective-C 오브젝트의 포인터를 저장하는데 사용함
- BOOL : bool 상수 값 (YES, NO)를 저장하는데 사용함

정적 오브젝트 선언은 해당 클래스의 오브젝트가 완전 선언이 되지 않는 경우에 사용 될 수도 있습니다. 이럴 경우 @class 키워드를 통해 컴파일러에게 현재 타입이 클래스임을 알려줄 수 있습니다.

```
@class NSArray, NSString, NSNumber;

@class NSDictionary;
```

위와 같이 선언한 NSArray, NSString, NSNumber, NSDictionary 는 정상적이고 유효한 클래스로서 정적타입선언이 가능합니다. 이러한 형태의 선언을 전방선언 (forward declaration)이라고 부릅니다. C 언어에서 구조체를 미리 선언해 두는 것과 같은 형식입니다.

Declaring a Class

클래스의 선언은 클래스의 인스턴스가 갖게될 값을 저장하는 것과 처리할 메시지를 명시하는 작업입니다. 클래스에서 처리할 메시지를 모두 기록하지 않아도 됩니다. 메시지는 고의로 감추거나 심지어는 runtime에 추가할 수 있습니다. (-_-?) 클래스 선언은 컴파일러가 메시지에 대한 힌트를 주는 정도지만 대신, 인스턴스내의 변수를 선언할수 있는 유일한 공간이기도 합니다.

한개의 클래스는 두 파트로 구성됩니다. “인터페이스” 와 “구현부” 입니다. 2장의 “Cocoa Language Options” 에서도 밝혔듯이 클래스는 데이터와 작동하는 기능을 캡슐화 하는데 사용됩니다. 클래스를 사용하는 곳에서는 구현부의 복잡성에 대해서는 감춰집니다.

몇가지의 새 키워드가 클래스 선언에 사용됩니다. 클래스 인터페이스는 @interface 로 시작되고 @end 로 끝납니다.

```
@interface MYObject : NSObject
{
    // nothing
```

```
}
@end
```

예제에서 MYObject 라는 새 클래스는 NSObject 의 하위클래스로 (subclass) 선언되었습니다. NSObject 는 Cocoa 의 한 부분이며, Cocoa 의 모든 클래스들은 직간접적으로 NSObject 의 서브클래스가 됩니다. NSObject 에 대한 정보는 이 장 후반부에 다시 다루어집니다.

현재 주목해야할 중요한 점은 MYObject 는 NSObject의 서브클래스이며 NSObject가 갖고 있는 변수들을 상속받고 NSObject 가 처리할 수 있는 메시지를 같이 처리할 수 있습니다. Objective-C 에선 다중상속(Multiple Inheritance)가 지원되지 않고 한개의 수퍼클래스(부모클래스)를 갖도록 선언할 수 있습니다. 새 클래스 선언시 콜론(:) 이하 부분을 생략하면 부모 클래스가 없어도 됩니다.

Instance Variable

MYObject 클래스 인터페이스에선 NSObject 에 정의된 것 외에 추가된 변수는 없습니다. 변수를 추가할 경우, 두 브레이스 ({ 와 }) 사이에 선언하면 됩니다. 변수가 추가될 경우의 일반적인 형태는 다음과 같습니다.

```
@interface CLASS-NAME : SUPER-CLASS-NAME
{
INSTANCE VARIABLE DECLARATIONS
}
METHOD DECLARATIONS
@end
```

인스턴스 변수는 기존에 선언된 타입이라면 어떤 것이라도 사용가능합니다. 예로서 , 아래 클래스는 가상으로 그리기(Hypothetical drawing)를 수행하는 원에 대한 것을 캡슐한 클래스 인터페이스 선언입니다.

```
@interface MYCircle : NSObject
{
    NSPoint  _myCenter;        // NSPoint는 cocoa의 C 구조체
    float    _myRadius;        // float는 C의 기본타입
    BOOL     *_myColor;        // NSColor는 Cocoa의 클래스
    id       _myExtraData;     // 어떤 오브젝트라도 대입 가능
}
@end
```

@public, @private, @protected 키워드를 사용해서 인스턴스 변수에 대한 접근 권한이나 범위를 제한할 수 있습니다.

public 인스턴스 변수는 어떤 코드에서도 사용가능합니다. @protected 인스턴스 변수는 이 변수를 선언한 클래스의 인스턴트와 해당 클래스의 서브클래스에서 사용이 가능합니다.

Private 인스턴스 변수는 자신이 속한 클래스에서만 사용이 가능합니다. 따로 권한을 명시하지 않는 다면 기본적으로 protected 권한을 갖게 됩니다. MYCircle 아래와 같이 접근권한을 나누어 변경되었습니다.

```
@interface MYCircle : NSObject
{
    NSPoint  _myCenter;
@public
    float    _myRadius;
@private
    BOOL     _myIsFilled;
@public
    NSColor  *_myColor;
    id       _myExtraData;
}
@end
```

한번 선언된 접근 권한 키워드는 다음 접근 권한 키워드가 선언 될때까지 유효합니다. MY-Circle 의 예제에서는, _myCenter 따로 선언된 것 없이 선언되었기 때문에 protected 입니다. _myRadius, _myColor, _myExternalData 인스턴스 변수는 public 의 권한을 갖습니다. _myIsFilled 인스턴스변수는 public 인스턴스 변수 중간에 선언되어 있지만, private 권한을 갖습니다.

Methods

Objective-C 에선 메시지를 받을 수 있도록 느슨하게 (어떤것이든) 선언됩니다. 같은 메시지라도 오브젝트에 따라 다르게 반응하게 할 수 있습니다. (당연한 이야기를) 다시 말하면 같은 메시지에 대해 각각 오브젝트들이 다른 메소드를 가질 수 있음을 의미합니다.

클래스를 선언할때마다 메시지에 반응하도록 메소드를 선언할 수 있습니다. 메소드는 처리할 메시지와 이름이 같아야 하며 오브젝트가 하는 일을 설명할때 사용되기도 합니다. 때문에 오브젝트의 메소드들과 오브젝트에서 처리하는 메시지는 일치하기도 합니다. “메소드 호출”은 “메시지 전송”으로 대치될 수도 있습니다.

클래스 인터페이스 선언시, 클래스에 구현된 메소드를 규정할 수 있습니다. 클래스의 메소드를 일부, 전체 또는 아예 없이 인터페이스 안에 선언할 수 있습니다. Static Typing 방식으로 클래스 메서드를 선언하면 컴파일러에게 도움이 됩니다. 하지만 이것들은 단지 힌트일 뿐입니다. 클래스 인터페이스에 선언되어 있지 않아도 구현부를 추가할 수 있으며 심지어 실행중(RunTime)에 추가할 수 있습니다.

메소드의 사용을 제한하기 위해 고의로 클래스 인터페이스에서 뺄 수 있습니다. Objective-C 에선 private, protected 로 메소드를 선언할 수 없습니다. 따라서 모든 메서드는 public입니다. 어쨋거나 특정 상황에서 메소드 호출을 막으려면 클래스 인터페이스에 아예 안넣

으면 됩니다. 만약 static typing 이 사용 되었다면 컴파일러는 선언되지 않은 함수를 사용했다고 “경고”를 냅니다. 이 경고의 의미는 호출되선 안될 함수가 호출되었다는 힌트입니다. (왜 이런 삽질을..) 메소드를 선언하는 테크닉, 컴파일러가 제대로 된 경고를 내도록,은 이 장의 후반부에 나옵니다.

메소드는 두 종류가 있습니다 : 인스턴스 메소드와 클래스 메소드. 인스턴스메소드는 클래스의 인스턴스가 메시지를 수신했을 때 호출됩니다. 클래스 메소드는 클래스에게 전달된 메시지를 수신했을 때 호출됩니다. 클래스 메소드는 Factory 메소드라고도 부르며, 이 의미는 클래스 메소드는 클래스 인스턴스를 만드는 역할을 많이 담당한다는 것입니다.

Note

각 클래스는 오브젝트로 실행시간(RunTime)에 구현됩니다. 클래스 오브젝트는 메시지를 수신할 수 있습니다. 클래스 오브젝트는 메타오브젝트(Meta-Object)라고 부르기도 하며 다른 오브젝트의 정보를 갖고 있기도 합니다. 클래스 오브젝트는 클래스 인스턴스의 선언부분을 캡슐화하며 인스턴스를 생성할때 사용합니다.

메소드 선언은 닫히는 브레이스 (}) 가 선언된 다음부터 @end 사이에 넣으면 됩니다.

```
@interface CLASS-NAME : SUPER-CLASS-NAME
{
    INSTANCE VARIABLE DECLARATIONS
}

METHOD DECLARATIONS

@end
```

인스턴스 메소드는 마이너스 부호 (-) 와 함께 선언됩니다.

```
- (int) count;
```

어떤 count 메시지라도 -count가 선언된 클래스의 인스턴스라면 메시지를 수신해서 -count메소드가 처리할 수 있습니다.

클래스 메소드는 플러스기호 (+) 와 함께 선언됩니다.

```
+ (void) setVersion:(int)number;
```

메시지 setVersion: 은 클래스 자신에게 전달되어 +setVersion: 에 의해 처리됩니다.

메소드의 리턴타입은 플러스나 마이너스 기호 다음에 선언되며 이건 마치 C 의 캐스팅 괄호 기호 같습니다. 어쨋거나 리턴값을 명시하는 것은 옵션입니다. 값을 명시하지 않으면 컴파일

러는 자동을 id 를 리턴값으로 간주합니다. 리턴값이 없는 경우에는 분명히 void를 적어줘야 합니다.

메소드 이름 다음의 메소드 이름과 메소드 확장부분은 세미콜론을 만나면 종결됩니다. 앞선 예제에서 , -count 메소드는 파라미터를 갖지 않습니다. +setVersion: 메소드는 한개의 정수형 파라미터를 받습니다. 입력 파라미터는 콜론(:) 다음에 위치합니다.

콜론 자체는 함수 이름의 일부입니다. 따라서 -init와 -init:는 다른 종류의 메서드로 받아들여집니다.

메소드의 이름 중 각 콜론 다음 부분의 괄호 안에 들어간 값은 입력 파라미터의 타입(type)이고 괄호 다음 부분은 변수 이름이 됩니다. 만약 파라미터에 타입이 명시되지 않으면 컴파일러는 타입을 id 로 간주합니다.

메서드 이름에 있는 콜론들은 파라미터에 이름을 부여합니다. 다음 메소드 선언을 읽어보십시요.

```
- (void) setArgument: (void*)argumentLocation atIndex: (int)index;
- (BOOL) lockWhenCondition: (int)condition beforeDate: (NSDate*)limit;
- (NSString*) descriptionWithCalendarFormat: (NSString*)format
    timeZone:(NSTimeZone*)aTimeZone locale:(NSDictionary*)locale;
```

이 메소드 선언들이 복잡해보이긴해도 실제로는 간단한 패턴을 따르고 있습니다. 각 파라미터의 뒷부분에서는 또 다른(추가되는) 메소드 이름, 콜론, 타입 선언, 변수이름을 기록할 공간이 있습니다. 이런 방식으로 파라미터는 무한정 추가될 수 있습니다.

(이상하죠, 더 읽어보기시 바랍니다.)

콜론을 함수 이름의 한 부분으로 간주해서 본다면 Objective-C 메소드 이름은 변수 이름 바로 앞부분 (타입이 적힌 괄호 바로 전의 스트링)에 해당하는 것들은 모두 해당됩니다. 그러므로 위의 3가지 예제에 대한 정확한 함수의 이름은 아래의 구문이 됩니다.

```
setArgument:atIndex:
lockWhenCondition:beforeDate:
descriptionWithCalendarFormat:timeZone:locale:
```

메소드의 이름을 변수 사이에 분산해서 쓸 수 있는데 , 이건 코드를 평범한 일반 문장처럼 읽어 내려갈 수 있게 해줍니다. 동시에 가독성, 유지보수, 명확성을 높입니다. 물론, 좋지 못한 이름을 선택한다면 이해하기 어려워질 것입니다. 임의로 메소드 이름의 구간을 재배치할 순 없습니다. 다음 두 함수의 이름은 전혀 다릅니다.

```
descriptionWithCalendarFormat:timeZone:locale:
descriptionWithCalendarFormat:locale:timeZone:
```

메소드 이름에서 콜론 사이를 빈공간으로 둘 수 있습니다. Objective-C 는 메소드 이름에 대해서 콜론만을 중요하게 처리하고 나머지는 생략할 수 있습니다. 하지만 메소드 설명부분을 활용하지 않는 좋지 못한 스타일입니다. 다음 함수는 정상적인 메소드입니다.

- (void) moveTo:(int)x :(int)y;

이와 동시에

moveTo::

가 선언됩니다. 정수값 x,y를 변수값으로 받고 리턴값이 없는 메소드.

메소드 이름은 자신이 처리할 메시지와 이름이 같습니다. 메소드는 메시지를 수신하면 호출됩니다. 메시지 이름으로 실행할 메소드를 선택합니다. 메시지 이름은 selector 라고도 부릅니다. 메시지의 이름은 (Objective-C 의 런타임에 의해) SEL 키워드로 정의된 변수에 저장 될 수 있습니다. 메소드 이름은 @selector 키워드를 사용해서 SEL 의 값으로 저장될 수 있습니다.

```
SEL           aSelector;
aSelector     = @selector( setObject:forKey: );
```

aSelector 의 값은 setObject:forKey: 메소드 이름을 가리키도록 설정되었습니다.

selector 는 메소드나 함수의 매개변수로 전달 할 수 있습니다. 예를 들어,

- (void) performSelector:(SEL)aSelector withObject:(id)anObject;

메소드는 anObject 의 aSelector 에 저장된 메소드를 호출할 수 있습니다. (또는 이 메소드를 호출하기 위해 위 메소드를 호출 할 수도 있습니다)

또한 특정 오브젝트에서 selector 가 사용할 함수의 포인터를 구해서 저장할 수 있습니다. 이 경우에는 IMP 라는 키워드로 선언된 변수에 저장합니다. IMP 는 최적화를 위한 소수의 경우에만 사용됩니다. IMP 를 사용하는 방법은 이 장에서 최적화 부분을 다루면서 다시 정리합니다.„

Implementing a Class

클래스의 구현은 @implementation 키워드로 시작되며 @end로 마감합니다. 클래스 구현부에는 메서드가 구현됩니다. 메소드 구현과정은 C 함수 구현과 같습니다. 메소드 이름 다음부터 구현내용이 시작되며 닫는 브레이스에서 종결됩니다. 다음 MYAverager 클래스를 보십시요:

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

@interface MYAverager : NSObject
```

```
{
    float        _myValueArray[10];
}

- (float)averageValue;
- (void)setValue:(float)aValue atIndex:(int)anIndex;

@end
```

MYAverager 는 간단한 클래스입니다. 열개의 부동소수점(floating-point)을 처리하며 -setValue:atIndex: 메소드로 어느 위치든 저장할 수 있습니다. 이 부동소수점들의 평균값은 -averageValue 메소드로 얻을 수 있습니다. 이 내용으로 MYAverage 클래스는 다음과 같이 구현됩니다.

```
#import “MYAverager.h”

- (float)averageValue
{
    int      i;
    float    sum = 0.0f;

    //sum the values
    for ( i = 0; i < 10; i ++ ) {
        sum = sum + _myValueArray[i];
    }

    return sum / 10.0f; //return the average
}

- (void)setValue:(float)aValue atIndex:(int)anIndex
{
    //set the value with the specified index
    if ( anIndex >= 0 && anIndex < 10 ) {
        _myValueArray[i] = aValue;
    }
}

@end
```

self and super

클래스를 구현하는 동안에는 자기 자신에게 메시지를 보내는 것이나 다른 곳(오브젝트)에 보내는 메시지에 현재 클래스를 파라미터로 사용하는 것이 때로는 도움이 됩니다. 이렇게 하기 위해 Objective-C에는 self 라는 숨겨진 키워드를 갖고 있습니다. 인스턴스 메소드에선 self

는 인스턴스 오브젝트의 포인터입니다. 클래스 메소드에서 self 는 메시지를 수신하는 클래스 오브젝트의 포인터가 됩니다.

The self variable can occur in any context that allows variables. 다음과 같이 메시지 수신자로 사용할 수 있습니다.

```
[self setValue:10.0f atIndex:4]
```

self의 값은 대입 가능하며 메소드의 리턴값이 될 수 도 있습니다. 종종 다른 메소드로 넘어가는 파라미터가 되기도 합니다.

Note

모든 Objective-C 메소드들은 호출된 때의 selector 를 저장한 _cmd 라는 숨겨진 (잘 사용되지 않는) 파라미터를 가지고 있다.

오브젝트에는 수퍼클래스(부모클래스)에 구현된 메소드를 다시 구현하기도 합니다. super 키워드는 수퍼클래스에 구현된 메서드를 호출할 때 사용할 수 있습니다. super 키워드는 변수가 아닙니다. 그리고 메시지를 수신하는 메서드에서만 사용 할 수 있습니다. 또한 메서드 구현부 안에서만 사용할 수 있습니다.

-init 메소드를 바로 전에 만든 MYAverager 클래스에서 추가해 보겠습니다. 다음 코드는 self 와 super 의 사용 방법에 대한 예제입니다.

```
- (id)init
{
    int      i;

    //set the self variable to the value returned from the
    // inherited implementation of -init
    self = [super init];

    //initialize the stored values by sending messages to self
    for ( i = 0; i < 10; i ++ ) {
        [self setValue:0.0f atIndex:i];
    }

    return self;
}
```

Creating Instances

클래스를 만든 뒤에 NSObjec의 클래스 메소드 +alloc 로 클래스의 인스턴스를 만들 수 있습니다. +alloc 를 상속받아야 하는 이유는 대부분의 클래스가 NSObject 로부터 서브클래싱되었기 때문입니다. 인스턴스가 생성된 뒤에는 인스턴스의 메소드로 초기화되어야 합니다. 본 장에 있는 NSObject 클래스 설명 부분에서는 인스턴스 생성 부분에 대한 내용을 다룹니다. 인스턴스에 대한 메모리할당(allocating)과 초기화의 프로세스는 컨벤션(convensions)이 담당합니다. 이 부분은 다음 장에서 설명합니다.

Summary

Objective-C 언어의 중요한 요소들이 한 장으로 설명이 가능했던 것은 언어의 단순성 덕분이다. Objective-C 의 runtime 에 대한 좀 더 자세한 정보는 부록 A 에서 찾아볼 수 있다. 이 부록에서 Objective-C의 강력한 기능들에 대해 수준 있는 테크닉을 배울 수 있다. 그러다보면 이 책의 내용으로 코코아 프로그래밍이 가능해질 수 있을 것임.

어떤 언어든 재사용이 가능한 코드의 라이브러리는 무척 중요하다. Objective-C 는 매우 잘 짜여진 언어이고 Cocas 프레임웍은 (전에 없이 훌륭한) 재사용이 가능한 오브젝트로 구성되어 있다. 라이브러리가 언어 활용도를 높여주는데 필요한 것 처럼 Convention 라이브러리의 활용도를 위해 필요하다. (convention이 뭐지?) Convention 에 필요한 것 중 새로운 것은 없다. Microsoft Windows 개발에 사용되는 MFC 에도 독자적인 Convention이 있으며 , 부정확한 사용을 막아준다. 전통적인 맥 라이브러리들은 파스칼 스타일의 스트링과 독자적인 메모리 관리에 대한 관용구가 있다. 5장에서는 Cocoa를 효과적으로 사용하기 위한 Cocoa 프레임웍의 관용구를 소개하겠다.