

Principles of Programming Languages

# Objective-C

Joris Kluivers  
[joris.kluivers@gmail.com](mailto:joris.kluivers@gmail.com)

<b>History .....</b>	<b>3</b>
<b>NeXT.....</b>	<b>3</b>
<b>Language Syntax.....</b>	<b>4</b>
<b>Defining a new class.....</b>	<b>4</b>
<b>Object identifiers .....</b>	<b>5</b>
<b>Sending messages.....</b>	<b>5</b>
<b>Strings.....</b>	<b>5</b>
<b>Protocols.....</b>	<b>6</b>
<b>Memory Management.....</b>	<b>6</b>
<b>Reference counting.....</b>	<b>6</b>
<b>Dynamic Typing.....</b>	<b>7</b>
<b>Error handling.....</b>	<b>8</b>
<b>Compiling your program.....</b>	<b>8</b>
<b>Like a scripting language.....</b>	<b>9</b>
<b>Resources.....</b>	<b>10</b>

## History

Until the 1980s most software development was based on structured programming, procedural programming using functions. The functions were used to break up larger programs into smaller parts to reuse code and make the program easier to work on. But as computers grew stronger and had more memory so did the software. Software became larger and more complicated, the use of functions just wasn't enough anymore to keep the code clean, organized and easy to maintain.

A trend became visible where software engineers thought Object Oriented Programming (OOP) was the solution to keep software maintainable. OOP languages existed already, and new ones were developed. Smalltalk for example was a language that already addressed the software engineering problems using OOP principles. Smalltalk needed a virtual machine to run though.

New languages were also developed. C++ added OOP programming features as an extension to C and quickly became the language of choice for native OOP development. C++ started with the addition of classes followed by among others, virtual functions, operator overloading, multiple inheritance and templates.

At the same time another extension of C was created called Objective-C. Objective-C was designed by Brad Cox at this company Stepstone. Cox created the language with true reusability in software design and programming in mind. He modeled the language with as many features from Smalltalk as possible. Not only did he create an OOP language, he also wrote a basic set of libraries and allowed the code and resources to be bundled in a single cross-platform format called universal binaries.

### **NeXT**

Steve Jobs, CEO and founder of Apple left the company in 1985 after an internal struggle. He analyzed the current the computer industry and formed a company called NeXT Inc. His product was what he thought would be the next big thing: an object-oriented toolkit, aimed at the academic market, using PostScript as the display technology. NeXT licensed Objective-C from StepStone. NeXT released its own compiler and libraries on which the NeXTStep OOP toolkit was based. In 1993 Next released NeXTStep for Intel and stopped producing hardware. NeXT changed it's name into NeXT Software Inc.

The GNU project released their own version of the ObjectiveC compiler and cloned the NeXTStep framework into GNUStep. The first gnu-objc runtime for GCC was released in 1992.

In 1996 Apple acquired NeXT and Steve Jobs once again became the CEO of the company he founded. NeXT executives mostly replaced their Apple counterparts. Apple ported NeXTStep to Apple PowerPC's and based it's new operating system Mac OS X on the NeXTStep framework. The NeXT step

framework was renamed Cocoa and is now the primary development framework for Apple software.

GNUStep for linux and windows also still exists but is maintained by a few volunteers and is not widely supported. GNUStep is kept in sync with the Cocoa framework but always runs behind the Apple development and never truly mirrors all Cocoa classes. But at the moment it's the only real solution to develop Objective-C on non Mac OS X systems.

## Language Syntax

Objective-C is a pure extension of C. This means all C code can be compiled using an Objective-C compiler. This can not be said of C++ which has some inconsistencies with C. Objective-C adds one data type, the object and one operation, the message expression to the C language. The message expression syntax is borrowed from the Smalltalk language. Some new keywords are also introduced.

### *Defining a new class*

The description of an object in Objective-C is given using an interface declaration:

```
@interface MyObject : ParentObject {
    // object variables
}
+ classMethod
+ classMethod

- instanceMethod
- instanceMethod
@end
```

MyObject is the name of the object defined here. Objective-C only allows single inheritance, in the example above, MyObject inherits from ParentObject. After the object variables block a list of methods is defined.

In your application a factory object is created for each class you defined. This is done at compile time and they are assembled in memory by the loader. One of the purposes of a factory object is to create initialized instances of their class. Since factory methods are real objects they can do anything the programmer defined. As a convention Factory object names always start with a uppercase letter and represent the class of the object they produce (like MyObject in the above example).

Messages send to factory objects are defined using a + sign. Messages send to the actual object instance are defined using a - sign.

The implementation for a defined interface is given using the following syntax:

```
@implementation
+ classMethod {
    // implementation
}
- instanceMethod {
    // implementation
}
@end
```

### ***Object identifiers***

In Objective-C objects can be identified in two separate ways, using a pointer to the object or using an object identifier.

```
int studentNumber; // declare an integer
Student *student; // pointer to an object
id student; // declare an object identifier
```

The student object pointer can only point to Student objects or instances of a subclass of Student. The object identifier can point to any object in Objective-C.

### ***Sending messages***

Messages can be seen as requests for operations on objects. As the programmer you send a message to the object, syntactically comparable with executing a method on an object in Java. The name of the message send to the object is called the selector.

Say you have an object identifier called student for a Student object, you send messages to the student as follows:

```
[student studentNumber]; // selector: studentNumber
[student setName:@"Joris Kluivers"]; // selector: setName:
[student setGrade:8 forClass:@"Data Structures"];
    // selector: setGrade:forClass:
```

As you can see, Objective-C messages use labeled parameters. Internally this might be translated to the c function `objc_setGrade_forClass(int i, NSString *name)`. The method parameters can't be rearranged like in python. The only function of the labeled parameters is self documentation.

### ***Strings***

Like Java supports a shorthand creation for string objects, so does Objective-C. When using the Objective-C string syntax `@"This is a string"` the compiler will create a NSString object. You can also supply your own class to be used at compile time by passing a special compiler flag.

## **Protocols**

Objective-C doesn't support multiple inheritance. At NeXT multiple inheritance by specification, not by implementation was added to Objective-C in the form of protocols. Protocols are much like Java interfaces, they define abstract methods to be implemented by the object that implements the protocol.

```
@protocol Locking
- (void) lock;
- (void) unlock;
@end
```

```
@interface MyClass : NSObject <Locking>
...
@end
```

## **Memory Management**

To actually get an object identifier you need to instantiate an object. This can be done using one of the factory methods. The factory methods for a Student class could be

```
id student = [Student new];
id student2 = [Student stringWithName:@"Joris Kluivers"];
```

Factory methods are responsible for allocating the object. In plain Objective-C this should be done using

```
+ new {
    id newObject = (* _alloc)(self, 0);
    return newObject;
}
```

statement. This looks quite complicated, fortunately the Cocoa framework defines two factory methods on the root object (called NSObject). The **new** method allocates a new instance and initializes all instance variables. A second method called **alloc** allocates the new instance but doesn't initialize the object. Usually a method called **init** is used to initialize the object.

```
id myObject = [MyObject new];
id myObject = [[MyObject alloc] init];
```

## **Reference counting**

Objective-C employs reference counting as memory management system. Each object keeps an internal count of the number of references to the object. A new object always starts with a reference count of 1. When you're done using the object you call the release method. The release method decreases the internal

reference count. When an object's reference count reaches 0 the **dealloc** method will be called on the object. The **dealloc** method should cleanup internal references and release objects it has references to.

When you use an object for a longer period of time you don't want it to be released by some other part of the program which leaves you with an invalidated object. To prevent an object from being released you send it the **retain** message. When an object receives a **retain** message it increases the internal reference count. This ensures the object won't be released until you send it a **release** message again.

Garbage collection is currently not a standard feature of Objective-C. Third party extensions are available and Apple is rumored to be working on a garbage collector for a future version of the Cocoa framework.

## Dynamic Typing

Like Smalltalk, Objective-C is a dynamically typed language. Messages can be sent to objects that weren't specified in the interface. This is where selectors come into play. A selector represents the name of the method, and only one selector exists for each method with the same name and argument labels. This way you can lookup the method using a string value and bind it at runtime. When an object receives a message it looks if it can find an implementation for the selector on the object itself other wise it will forward the selector to one of its super classes. The programmer can also decide to capture the message manually and send it off to a totally different object. In case the message can not be forwarded to another object an error handler will be used to throw a runtime exception.

Sending the wrong message to the wrong object will cause run-time errors. To prevent these errors at runtime you could specify the class of an object, so the compiler will apply static-tying and will complain at compile time about messages an object will not respond to.

- (void) setMyValue:(id)someObject;
- (void) setMyValueToAString:(NSString \*)aString;

Internally functions and messages differ in several ways. The most important difference is the parameter count. Although this is not visible in the source code, messages always have one parameter that identifies the object the message is sent to. This object can be used in a method using the name self.

Because of its dynamic nature Objective-C has some really nice features usually only to be found in scripting languages. My favorite is the category. To keep your code modular Objective-C enables you to extend objects at runtime. For example to extend a NSString object to include a method that encodes it for url usage you could create a category called WebExtensions:

```

@interface NSString (WebExtensions)
- (NSString *) urlEncoded;
@end

@implementation NSString (WebExtensions)
- (NSString *) urlEncoded {
    // implementation
}
@end

```

The advantage of using this over class extensions is you don't have to alter your code to use your extended class. The urlEncoded method will be added to all NSString's. Even if these NSStrings are created in a third party library and returned to you. The only languages I know of that support a feature like this are scripting languages like Ruby.

## Error handling

Objective-C uses different ways of handling errors. When using C libraries like the Carbon (OS X c libraries) framework most of the error handling is done using return codes. Methods will return special codes for success or failure.

Another method of handling errors is using the NSError object. A pointer to NSError is passed as an argument in a message. The NSError will be nil if the message succeeded or contain information about the error when one occurred.

A third way of handling errors is using exceptions. Exception handling is very similar to Java exception handling. By default exception handling is done using the macro's NS\_DURING and NS\_HANDLER. A special compiler option enables the use of the compiler directives @try, @catch and @finally for a more Java like experience.

```

@try {
    // some code that might throw an exception
} @catch (AnObjCClass *exc) {
    // handle exception
} @finally {
    // cleanup
}

```

Throwing an exception can be done using the **raise:format:** method on a NSException object, or the shorthand **@throw ExceptionObject**.

## Compiling your program

Good practice is to define object definitions in header files (.h) and the object implementations in implementation files (.m) like the C (.h, .c) files. But if you like you could have all code in one file.

A beginning developer wouldn't know how to compile objective-c code by hand. All compiling is handled by the developer tools (usually XCode). Under the hood XCode is using GCC to compile the code. Compiling Objective-C code using GCC requires a special compiler flag and requires you to link to the objc runtime. This runtime is very small and won't add much to the size of the application. And by using the runtime you save the space of an entire virtual machine.

## **Like a scripting language**

Objective-C offers everything C has to offer, but adds features which makes Objective-C very dynamic. Because of this Objective-C behaves very much like a scripting language with the speed of a native C language. Writing code in Objective-C tends to be very clean and although the messaging syntax is something you might need to get used to it feels very natural after a while.

One disadvantage of the Objective-C language is its lack of support for modules or packages. You can't just name your classes and a common practice is to prefix your class names using one or two characters. The NS prefix is preserved to Apple, inherited from the NeXTStep era.

Apple makes use of these dynamic features (some of which it added to Objective-C itself) a lot in it's development tools to move the load from the developer to the development tools resulting in writing less code. This make writing applications for OS X very easy and fast.

# Resources

## Books

Brad J. Cox and Andrew J. Novobilski: “ Object-Oriented Programming: An evolutionary approach, Second Edition”, 1991

## Online Documents

<http://en.wikipedia.org/wiki/Objective-C>

<http://en.wikipedia.org/wiki/NeXT>

<http://developer.apple.com/>