

01: Objective-C

Objective-C, Metódus hívások, Egymásba skatulyázott üzenetek, Több bemenettel rendelkező metódusok, Accessor-metódusok, A pont szintaktika

Learn Objective-C

Objective-C

Az Objective-C az elsődleges programozási nyelv amikor Mac szoftvert szeretnénk írni. Ha gyakorlott vagy a C programozásban és ismered az objektum orientált programozás alapjait, akkor nem jelent majd gondot az Objective-C megértése. Amennyiben nem ismered a C programozási nyelvet, akkor feltétlenül kezd először a [Learn C cikk](#) elolvasásával.

A cikk forrása: [Scott Stevenson](#):

[Cocoa Dev Central: Learn Objective-C](#)

Metódus hívások

A gyors kezdés érdekében nézzünk néhány egyszerű példát.

Egy objektum metódusát a következőképpen hívhatjuk meg:

```
[object method];  
[object methodWithInput:input];
```

A metódusoknak lehet visszatérési értéke is:

```
output = [object methodWithOutput];  
output = [object methodWithInputAndOutput:input];
```

Metódusokat osztályokon is meg lehet hívni és így lehet objektumot készíteni. Az alábbi példában meghívjuk a **string** metódust az NSString osztályon és egy új NSString objektumot kapunk eredményül:

```
id myObject = [NSString string];
```

Az `id` típus azt jelöli, hogy a `myObject` változó bármilyen típusú objektum lehet, tehát az alkalmazás fordításakor még nem ismert az aktuális osztály és a metódus típusa.

A fenti példában nyilvánvaló, hogy az objektum típusa `NSString` lesz, ezért konkretizálhatjuk a típusát:

```
NSString* myString = [NSString string];
```

Ez most már egy `NSString` típusú változó lesz, ezért a fordító program figyelmeztetni fog minket, ha egy olyan metódust próbálunk használni ezen az objektumon, amelyiket az `NSString` nem támogat.

Vegyük észre, hogy az objektum típusának jobb oldalán van egy csillag karakter. Mindegyik Objective-C objektum változója pointer típusú. Az `id` típus eleve pointer típusúnak van definiálva, ezért nem kell mellé kitenni a csillagot.

Egymásba skatulyázott üzenetek

A legtöbb programozási nyelvben így néznek ki az egymásba skatulyázott metódusok, vagy függvény hívások:

```
function1 ( function2() );
```

A `function2` eredménye lesz a `function1` bemenete. Az Objective-C-ben az egymásba skatulyázás így néz ki:

```
[NSString stringWithFormat:[prefs format]];
```

Célszerű mellőzni a kettőnél több egymásba skatulyázást egy program soron belül, mert az könnyen az áttekinthetőség rovására mehet.

Több bemenettel rendelkező metódusok

Néhány metódusnak több bemenete is van. Az Objective-C-ben a metódus nevét több szegmensre is oszthatjuk. A header-ben a több bemenettel rendelkező metódus így néz ki:

```
-(BOOL)writeToFile:(NSString *)path atomically:(BOOL)useAuxiliaryFile;
```

Ezt a metódust így lehet meghívni:

```
BOOL result = [myData writeToFile:@"tmp/log.txt" atomically:NO];
```

Itt nem csak címkézett paraméterekről van szó. A *writeToFile:atomically:* a metódus neve a futtatás során.

Accessor-metódusok

Mivel az Objective-C-ben a példányváltozók alapból private változók, ezért a legtöbb esetben accessorokat (hozzáférőket) kell használni az értékek lekéréséhez, beállításához (getter, setter metódusok). Kétfajta formátuma is létezik ennek. Az egyik a hagyományos 1.x szintakszis:

```
[photo setCaption:@"Day at the Beach"];  
output = [photo caption];
```

A második sorban nem kapjuk meg közvetlenül a példányváltozó értékét, hanem meghívjuk a **caption** metódust. Az Objective-C-ben a legtöbb esetben nem tesszük ki a "get" előtagot. Amikor szögletes zárójellel találkozunk a programkódban, akkor az mindig azt jelenti, hogy egy üzenetet küldünk egy objektumnak, vagy egy osztálynak.

A pont szintaktika

Az értékek lekéréséhez, beállításához a pont sziktatikát (dot syntax) is használhatjuk. Ez a lehetőség az Objective-C 2.0 verziótól él, ami a Mac OS X 10.5 része:

```
photo.caption = @"Day at the Beach";  
output = photo.caption;
```

Mindkét formátum használható, de egy projekten belül csak az egyik fajtát célszerű alkalmazni. Fontos megjegyezni, hogy a pont sziktatika csak az adatok lekéréshez, beállításához használható, nem egy általános metódus szintaktika.

02: Objective-C

Objektum készítés, Alapvető memória kezelés

Objektum készítés

Két alapvető lehetőség is van objektum készítésre. Az elsőt már láttuk az imént:



```
NSString* myString = [NSString string];
```

Ez a kényelmesebb, automatikus változat. Ebben az esetben egy **autorelease** objektumot készítünk, aminek tárgyalására még a későbbiekben részletesen visszatérünk. Azonban sok esetben a manuális módszert kell választanunk:

```
NSString* myString = [[NSString alloc] init];
```

Ez egy egymásba skatulyázott metódus hívás. Az első lépésben az NSStringen meghívjuk az **alloc** metódust. Ez egy relatíve alacsony szintű metódus hívás, amelyik lefoglalja a szükséges memória területet és létrehozza az objektum példányt.

A második lépésben meghívjuk az **init** metódust az új objektumon. Az **init** végrehajtása rendszerint elvégzi az alapbeállításokat, például elkészíti a változó példányokat. Ennek a részletei nem láthatók számunkra, mivel ennek az osztálynak csak használói vagyunk.

Más esetekben egy másik változatot használunk, amikor az **init**-nek kimenete is van:

```
NSNumber* value = [[NSNumber alloc] initWithFloat:1.0];
```

Alapvető memória kezelés



alloc⁺¹

Amikor Mac OS X-re írunk egy programot, lehetőségünk van bekapcsolni a garbage collection-t (szemétygyűjtést).

Ez azt jelenti, hogy a komplikált esetektől eltekintve nem kell foglalkozni a memória kezeléssel. Azonban vannak olyan programkörnyezetek, ahol nem használható a garbage collection (ilyen pl. az iPhone OS jelenleg). Emiatt feltétlenül jó ismerni néhány alapvető koncepciót.

Amennyiben manuálisan létrehozunk egy objektumot az **alloc** metódussal, akkor később ezt feltétlenül követnie kell egy **release** metódusnak, azaz fel kell szabadítani az objektumot.

Azonban nem szabad manuálisan felszabadítani egy autorelease-zel létrehozott objektumot, mert ebben az esetben az alkalmazás le fog fagyni.

Lássunk két példát:

```
// string1 automatikusan fel lesz szabadítva  
NSString* string1 = [NSString string];
```

```
// manuálisan kell felszabadítani, amikor már nincs rá szükség  
NSString* string2 = [[NSString alloc] init];  
[string2 release];
```

Első közelítésben nyugodtan feltételezhetjük, hogy egy automatikus objektum fel lesz szabadítva amikor az aktuális függvény befejezte a működését.

Sokat kell még tanulnunk a memória kezelésről, de ennek akkor lesz majd értelme amikor több ismerettel rendelkezünk.

03: Objective-C

Az osztály interfész megtervezése, Metódusok megadása, Osztály implementáció, Init, Dealloc

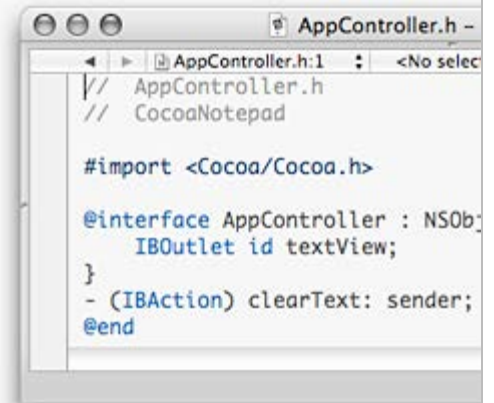
Az osztály interfész megtervezése

Az Objective-C-ben nagyon egyszerűen tudunk osztályokat létrehozni. Ez alapvetően két részből áll.

Az osztály interfész rendszerint a **ClassName.h** fájlban található, a változó példányokat és a publikus metódusokat definiálja.

Az implementáció pedig a **ClassName.m** fájlban található és a fenti metódusok aktuális program kódját tartalmazza. Gyakran tartalmaz privát metódusokat is, amelyek az osztály kliensei számára nem elérhetők.

Lássunk egy példát arra, hogy néz ki egy ilyen interfész. Az osztály neve a Photo, ezért a megfelelő fájl neve **Photo.h**:



```
AppController.h -
AppController.h:1
// AppController.h
// CocoaNotepad

#import <Cocoa/Cocoa.h>

@interface AppController : NSObject {
    IBOutlet id textView;
}
- (IBAction) clearText: sender;
@end
```

```
#import

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
@end
```

Először is beimportáljuk a **Cocoa.h-t** annak érdekében, hogy az alapvető osztályokat elérhetővé tegyük a Cocoa alkalmazás számára. Az **#import** direktíva automatikusan kizárja, hogy ugyanazt a fájlt többször is beimportáljuk.

Az **@interface** azt mutatja, hogy ez a **Photo** osztály egy deklarációja. A kettőspont a superclassra (főosztály) utal, ami jelen esetben az **NSObject**.

A kapcsos zárójeleken belül található két változó példány: **caption** és **photographer**.

Most éppen mindkettő **NSStrings**, de bármilyen objektum típusok is lehetnének, akár **id** típusok is.

Végezetül a **@end** szimbólum lezárja az osztály definícióját.

Metódusok megadása

Adjunk meg néhány lekérdező metódust a példányváltozókhöz:

```
#import

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}

- caption;
- photographer;

@end
```

Emlékeztetünk, hogy az Objective-C metódusaiban jellemzően elhagyjuk a "get" előtagot. A metódus neve előtt levő kötőjel azt jelenti, hogy ez egy példány metódus. A plusz jel pedig azt jelöli, hogy ez egy osztály metódus. Ha nem mondunk mást, akkor a fordító program azt feltételezi, hogy a metódus visszatérési értéke az id, és az összes bemenő érték típusa is id. A fenti program kód technikailag ugyan korrekt, de egy kicsit furcsa. Jobb ha megadjuk a visszatérési értékek típusát is:

```
#import

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}

- (NSString*) caption;
- (NSString*) photographer;

@end
```

Most készítsük el a setter metódusokat:

```
#import

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}

- (NSString*) caption;
- (NSString*) photographer;
```

```
- (void) setCaption: (NSString*)input;
- (void) setPhotographer: (NSString*)input;
```

```
@end
```

A setter metódusoknál nem szükséges, hogy legyen visszatérési érték, ezért nyugodtan definiálhatjuk őket **void**-nak.

Osztály implementáció



Lássunk egy osztály implementációt, kezdjük a getter metódusokkal:

```
#import "Photo.h"

@implementation Photo

- (NSString*) caption {
return caption;
}

- (NSString*) photographer {
return photographer;
}

@end
```

A megfelelő kódrészlet **@implementation** -nal és az osztály nevével kezdődik és **@end** -del végződik, ugyanúgy, ahogy az interfész. Az összes metódust ezen két utasítás közé kell tenni. Ha már programoztál valamikor, akkor a getter metódusok könnyen érthetők lesznek számodra, ezért inkább a setter metódusokra fogunk most több figyelmet fordítani:

```
- (void) setCaption: (NSString*)input
{
[caption autorelease];
caption = [input retain];
}

- (void) setPhotographer: (NSString*)input
{
[photographer autorelease];
```

```
photographer = [input retain];  
}
```

Mindegyik setter két változóval dolgozik. Az első egy létező objektumra való utalás, a második az új input objektum. Ha a garbage collection használható, akkor az új értéket közvetlenül is beállíthatjuk:

```
- (void) setCaption: (NSString*)input {  
caption = input;  
}
```

Ha viszont a garbage collection nincs bekapcsolva, akkor a régi objektumot fel kell szabadítani (**release**) és meg kell tartani (**retain**) az újat.

Kétféleképpen szabadíthatunk fel egy objektumra való hivatkozást: **release** és **autorelease**. A szabványos release azonnal kitörli a hivatkozást. Az autorelease metódus egy késleltetett törlést jelent, de jellemzően megvárja az aktuális függvény végrehajtásának a befejezését (feltéve, hogy speciális utasításokkal ezt nem írjuk felül).

Az autorelease metódus biztonságosabb a setter-en belül, mivel a régi és az új értékeket tartalmazó változók mutathatnak ugyanarra az objektumra. tehát nem biztos, hogy azonnal törölni szeretnél egy objektumot amit éppen szeretnél megtartani.

Ebben a pillanatban ez kicsit zavaros lehet, de a gyakorlat munka során majd érthetőbb lesz, nem gond ha most még nem teljesen érthető minden részlet.

Init

Készíthetünk egy init metódust annak értelmében, hogy beállítsuk a példányváltozónk kezdeti értékét:

```
- (id) init  
{  
if ( self = [super init] )  
{  
[self setCaption:@"Default Caption"];  
[self setPhotographer:@"Default Photographer"];  
}  
return self;  
}
```

A második sortól eltekintve teljesen magától érthetődő ez a kód. A második sorban pedig egy egyszerű egynelőség jel látható ami a **self** -hez hozzárendeli a `[super init]` értéket. Ennek az az értelme, hogy megkéri a superclass-t, hogy végezze el a saját inicializálását. Az **if** utasítás ellenőrzi, hogy az inicializálás sikeres volt-e mielőtt a kezdeti értékeket beállítjuk.

Dealloc

A **dealloc** metódust kell meghívunk egy objektumon amikor azt szeretnénk kitörölni a memóriából. Rendszerint ez a legjobb pillanat arra, hogy töröljük az összes gyerek példányváltozóra mutató hivatkozást:

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

Az első két sorban elküldjük a `release` hívást az összes példányváltozónak. Nem kell az `autorelease` hívást használnunk most, és a szabványos `release` még egy kicsit gyorsabb is. Az utolsó sorban a `[super dealloc]` üzenet nagyon fontos annak érdekében, hogy a superclass eltakarítsa saját magát. Amennyiben ezt nem tesszük meg, akkor az objektum nem lesz kitörölve aminek pedig memóriaelszivárgás lesz az eredménye. Amennyiben a garbage collection be van kapcsolva, akkor nincs szükség a `dealloc` metódus meghívására, helyette a **finalize** metódust kell implementálni.

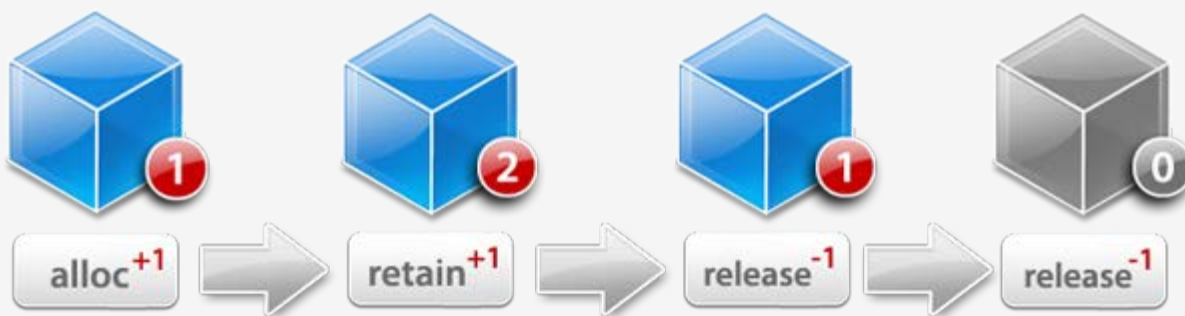
04: Objective-C

Memória kezelés folytatása, A naplózás

Memória kezelés folytatása

Az Objective-C memória kezelő rendszerét referencia számlálásnak nevezzük. Mindössze annyi a dolgunk, hogy figyelni kell a hivatkozásaink referencia számát és futás közben tisztán kell tartani a memóriát.

Leegyszerűsítve ez annyit jelent, hogy **lefoglalod egy objektum (alloc)** helyét a memóriában, esetleg **megőrzöd azt (retain)** a program egy másik pontján, vagy pontjain, aztán küldesz egy **felszabadítást (release)** mindegyik elküldött alloc/retain utasításra. Ez például azt jelenti, hogy ha használtál egy alloc-ot és később egy retain-t, akkor kétszer ki kell küldeni a release-t.



Ez a referencia számlálás elmélete. Gyakorlatilag azonban csupán két okból hozunk létre objektumot:

1. Példányváltozóként szeretnénk használni
2. Csak ideiglenesen szeretnénk használni azt egy függvény belsejében

A legtöbb esetben egy példányváltozó setter módszerának **fel kell szabadítania** a régi objektumot és **meg kell tartania** az újat. Ezután még fel kell őt szabadítani a dealloc módszerban is.

Így valójában csak arra kell figyelniünk, hogy egy függvényen belül a lokális hivatkozásokat megfelelően kezeljük. Itt pedig egy szabály van: ha készítesz egy objektumot az **alloc**, vagy a **copy** módszerrel, akkor el kell küldeni a **release**, vagy az **autorelease** üzenetet a függvény végén. Ha más módon készítesz egy objektumot, akkor nincs semmi további teendő.

Az első esetre mutatunk most egy példát, a példányváltozó kezelésére:

```
- (void) setTotalAmount: (NSNumber*)input  
{
```

```
[totalAmount autorelease];
totalAmount = [input retain];
}

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

Ez pedig a másik eset, a lokális hivatkozásokkal. Csak az `alloc`-kal készített objektumot kell felszabadítanunk:

```
NSNumber* value1 = [[NSNumber alloc] initWithFloat:8.75];
NSNumber* value2 = [NSNumber numberWithFloat:14.78];

// csak a value1-et szabadítjuk fel, a value2-t nem
[value1 release];
```

A következő pedig egy kombinált eset, ahol a lokális hivatkozás segítségével állítjuk be az objektum példányváltozót:

```
NSNumber* value1 = [[NSNumber alloc] initWithFloat:8.75];
[self setTotal:value1];

NSNumber* value2 = [NSNumber numberWithFloat:14.78];
[self setTotal:value2];

[value1 release];
```

Megjegyezzük, hogy a lokális referenciák kezelése pontosan ugyanolyan, függetlenül attól, hogy példányváltozóként állítjuk elő őket, vagy pedig nem. Nem szükséges a setter implementációkon gondolkodni.

Amennyiben ezt megértetted, akkor legalább a 90%-át birtoklod annak a tudásnak, amire valaha szükséged lesz az Objective-C memória kezelésével kapcsolatban.

Az Objective-C-ben nagyon egyszerűen tudunk üzenetet küldeni a console-ra. Az `NSLog()` függvény majdnem tökéletesen megegyezik a C `printf()` függvényével, annyi különbséggel, hogy ez tartalmaz egy kiegészítő `%@` jelölést az objektumok számára.

```
NSLog ( @"Az aktuális dátum és idő: %@", [NSDate date] );
```

Így irathatjuk ki egy objektum tartalmát a console-ra. Az `NSLog` függvény meghívja a **description** metódust az objektumon, és kiírja a visszatört NSString értéket. Lehetőség van a `description` metódus testre szabására az aktuális osztályban és így a kívánságunknak megfelelő stringet kapjuk eredményül.

05: Objective-C

Tulajdonságok (Properties), Metódusok üres hívása, Kategóriák

Tulajdonságok (Properties)

Bizonyára észrevetted azokat az általános összefüggéseket az iménti programkódban, (amikor a `caption` és `photographer` objektumokhoz készítettünk accessor metódusokat,) amelyek alapján egyszerűbbé tehetjük a programunkat.

A property az Objective-C olyan jellemzője, ami lehetővé teszi accessor-ok automatikus generálását és ezen kívül még további előnyökkel is rendelkezik. Nézzünk egy példát erre, és konvertáljuk át a Photo osztályt a property segítségével.

Ez volt az eredeti kód:

```
#import

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
- (NSString*) caption;
- (NSString*) photographer;

- (void) setCaption: (NSString*)input;
- (void) setPhotographer: (NSString*)input;

@end
```

Ezt kapjuk a konvertálás után:

```
#import

@interface Photo : NSObject {
    NSString* caption;
    NSString* photographer;
}
@property (retain) NSString* caption;
@property (retain) NSString* photographer;

@end
```

A `@property` egy Objective-C direktíva amelyik a property-t definiálja. A zárójelek között látható "retain" meghatározza, hogy a setter-nek **meg kell tartania** az input értéket, és a sor hátra levő része egyszerűen meghatározza a property nevét és típusát.

Most vessünk egy pillantást az osztály implementációjára:

```
#import "Photo.h"

@implementation Photo

@synthesize caption;
@synthesize photographer;

- (void) dealloc
{
    [caption release];
    [photographer release];
    [super dealloc];
}

@end
```

A `@synthesize` direktíva automatikusan generálja a getter és a setter metódusokat, ezért egyedül a `dealloc` metódust kell elkészítenünk ehhez az osztályhoz.

Az accessor-ok csak akkor lesznek legenerálva, ha még nem léteznek, ezért nyugodtan használd a `@synthesize` direktívát a property-hez, azután készítsd el a saját getter és setter metódusaidat, amikre szükséged van. A fordító a hiányzó metódusokat pótolni fogja.

A property deklarációnak nagyon sok további opciója van, amelyek azonban túlmutatnak ennek a cikknek a keretein.

Metódusok üres hívása

Az Objective-C-ben a *nil* objektum funkcionálisan megegyezik a más programozási nyelvekben használt *NULL* pointerrel. Csupán annyi a különbség, hogy itt a metódusok nil hívása megengedett, nem okoz fagyást vagy más problémát.

Ezzel a technikával gyakran találkozunk a framework-ökben, de a legfontosabb előnye, hogy általában nem kell ellenőrizni a nil értéket mielőtt meghívjuk a metódust egy objektumon. Ha egy nil objektumon hívunk meg egy metódust aminek a visszatérési értéke egy objektum, akkor a visszatérési érték is a nil lesz.

Egy kissé továbbfejleszthetjük ezzel a `dealloc` metódusunkat:

```
- (void) dealloc
{
self.caption = nil;
self.photographer = nil;
[super dealloc];
}
```

Ez működik, mert amikor beállítjuk a `nil`-t egy példányváltozóként, a setter megtartja a `nil`-t (ami nem csinál semmit) és felszabadítja a régi értéket. Ez a megoldás gyakran jobb a `dealloc` számára mivel kizárja azt a lehetőséget, hogy a változó egy olyan véletlen adatra mutasson, amit egy objektum már használt.

Figyeljük meg, hogy itt a *self*. szintaktikát használtuk, ami azt jelenti, hogy a setter-t használjuk és biztosított számunkra a korrekt memória kezelés. Amennyiben közvetlenül használnánk az értékadást, például ahogy a következő példában látható, az memóriaelszívargást okozna:

```
// Hibás! Memóriaelszívargást okoz. // Használjuk a self.caption-t a setter használatának
érdekében
caption = nil;
```

Kategóriák

Az Objective-C egyik leghasznosabb tulajdonsága a kategóriák használata. A kategória lehetővé teszi, hogy metódusokat anélkül rendeljünk egy létező osztályhoz, hogy az annak az alosztálya lenne, vagy bármit is tudnánk az implementációjáról.

Ez különösen azért hasznos, mert beépített objektumokhoz is hozzá tudunk adni metódusokat. Ha például hozzá szeretnénk adni egy metódust az `NSString` összes példányához az adott alkalmazásban, akkor egyszerűen egy kategóriát kell hozzáadni. Nem kell mindent összszedni ami egy szokásos alosztályhoz szükséges.

Például, ha szeretnénk egy metódust hozzáadni az `NSString`-hez annak érdekében, hogy ellenőrizzük, hogy a tartalma egy URL-e, az így nézhet ki:

```
#import

@interface NSString (Utilities)
- (BOOL) isURL;
@end
```

Ez nagyon hasonló az osztály deklarációhoz, azzal a különbséggel, hogy nincs szuper osztály megnevezve, valamint a kategória neve megtalálható a zárójelek között. A kategória neve bármi lehet, mivel csupán a metódusok belső kommunikációjához használjuk.

Az implementáció pedig így néz ki (Vegyük figyelembe, hogy ez nem egy tökéletes megoldás az URL ellenőrzésre, ez csak egy példa a kategóriák bemutatására):

```
#import "NSString-Utilities.h"

@implementation NSString (Utilities)

- (BOOL) isURL
{
    if ( [self hasPrefix:@"http://"] )
        return YES;
    else
        return NO;
}

@end
```

Most már bármelyik NSString-re alkalmazhatjuk ezt a metódust. A következő kód ki fogja írni a konzolra, hogy "a string1 egy URL":

```
NSString* string1 = @"http://pixar.com/";
NSString* string2 = @"Pixar";

if ( [string1 isURL] )
    NSLog (@"a string1 egy URL");

if ( [string2 isURL] )
    NSLog (@"a string2 egy URL");
```

Ellentétben az alosztályokkal, a kategóriákhoz nem tudunk állapotváltozókat hozzáadni.

Azonban a kategóriák segítségével felül lehet írni egy osztály létező metódusait. Ezt viszont nagyon óvatosan kell megtenni.

Emlékezzél rá, hogy ha egy kategória segítségével megváltoztatsz egy osztályt az hatással lesz az osztály összes példányára az adott alkalmazáson belül.

Utószó

Ez a cikk az Objective-C egy rövid bevezetője. Ahogy láthattad, ez a nyelv gyorsan megtanulható, nem kell sok speciális szintaktikával bajlódni, és a Cocoa használata közben állandóan ugyanazokat a szabályokat alkalmazzuk.

Amennyiben működés közben is szeretnéd látni ezeket a példa programokat, töltsd le az alábbi projektet és tanulmányozd a forráskódot:

[LearnObjectiveC](#) Xcode 3.0 Project (56k)

Fordította: Major Zoltán

PDF: alkamzasfejlesztes.blogspot.com