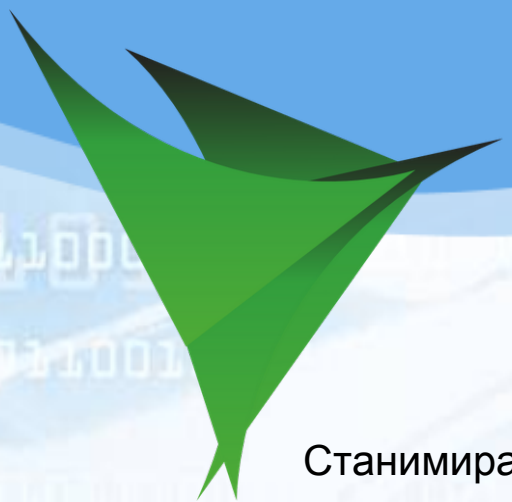
LOG
O

Въведение в Objective C



Станимира Железова

Съдържание

Objective C

- Обектно ориентиран език за програмиране
  - разширение на C
    - Всеки C код ще се компилира от Objective-C компилатор. Също може свободно да добавяте C код в Objective-C класове
  - базиран на пращане на съобщения от Smalltalk
    - Динамично свързване (всеки обект се достъпва през референция и всички функции са „виртуални“)
    - Типовете не се проверяват по време на компилация и получателят може да не отговаря на пращаното съобщение

Objective - C / C

- Псевдо-множествено наследяване (чрез препращане на съобщения и протоколи)
  - - (void)forwardInvocation:(NSInvocation *)anInvocation
  - @protocol – деклариране на методи, на които обектите отговарят
  - nil vs NULL - Получателят съобщение може да бъде nil и това няма да даде грешка runtime.
- BOOL – въведен е булев тип със стойности YES/NO (в хедъра на NSObject от Foundation)
- Има и някои липсващи неща:
  - No namespaces
  - No operator overloading

Основни парадигми

- Основен шаблон за дизайн - MVC
  - Model – променливи за състояние и data sourcing
  - View – Всички наследници на UIView и UIViewController
  - Controller
    - Delegation
    - Target-Action
    - Notifications
- Файлове в проект
    - .m, .h, .xib
    - main.m

Hello world

- main.m:

```
#import <stdio.h>
int main( int argc, const char *argv[] ) {
printf( "hello world\n" );
return 0;
}
```

  - Една Objective-C програма започва от main.m
  - Използваме #import вместо #include – подсигурява, че един header се добавя точно веднъж

Основен синтаксис

- **Основни типове:**

	Стойности	форматиране
char	‘a’, ‘\n’	%c
int	17, -99, 0xFFAE, 0878	%i, %x, %o
float	12.3f, 3.1e-5f, 0×1.5p10, 0×1P-1	%f, %e, %g, %a
double	12.34, 3.1e-5, 0×.1p3	%f, %e, %g, %a
BOOL	YES, NO	%d (prints 1 or 0)
id	nil, object refs	%p

```
@import <limits.h>
// ...
NSLog(@"CHAR_MIN: %c", CHAR_MIN);

// signed short int
NSLog(@"SHRT_MAX: %hi", SHRT_MAX);

NSLog(@"INT_MIN: %i", INT_MIN);
NSLog(@"LONG_MIN: %li", LONG_MIN);

// unsigned long int
NSLog(@"ULONG_MIN not defined, it's always
zero: %lu", 0););

// signed long long int
NSLog(@"LLONG_MIN: %lli", LLONG_MIN
NSLog(@"LLONG_MAX: %lli", LLONG_MAX);

// unsigned long long int
NSLog(@"ULLONG_MIN not defined, it's always
zero: %llu", 0);
```

Основен синтаксис

- void* vs id
  - void* се използва за указатели към неупоменат тип, а id е указател към обект, чийто клас не е упоменат (подобно на NSObject*)
  - id се предпочита винаги, когато знаем, че работим с обекти

    ```
    int someComparator(id obj1, id obj2, void *context) { ... }
    ```
- Тип SEL – указател към selector (съобщение, което може да бъде пратено към обект)

  ```
  SEL callback = @selector(methodWithParam:andParam2:);
  ```

Основен синтаксис

- **Функции**
  - **деклариране:**
    - **int add (int a, int b); //other language**
    - **-(int) addIntA: (int) AndIntB: (int); //Objective-C**
  - **извикване:**

    this.add(5, 2) //other language

    [self addIntA:5 AndIntB:2]; //Objective-C

- **Променливи**
  - **public BOOL isIt = NO;**
  - **private NSString *aStr = [NSString string];**
  - **NSNumber* value = [[NSNumber alloc] initWithFloat:**

Извикване на методи

- Базов синтаксис:

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

- Върната стойност :

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

- Конструктор :

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

Паралел с C++

- В C++ това :

  object.method(parameter);

- изглежда в Objective-C така :

  [object method: parameter]

Класове и обекти

- **Клас се декларира в @interface блок:**

```
// ---- @interface ----
@interface Fraction : NSObject { // inherit from NSObject
int numerator;
int denominator;
}
- (void)print;
- (void)setNumerator:(int)n;
- (void)setDenominator:(int)d;
@end
```

- **Клас се имплементира в @implementation блок**

```
@implementation Fraction // you can optionally specify :NSObject after Fraction
- (void)print { NSLog(@"%i/%i", numerator, denominator); }
- (void)setNumerator:(int)n { numerator = n; }
- (void)setDenominator:(int)d { denominator = d; }
```

Паралел със C++

в C++ пишем... :

```
class classname : superclassname {
public:
  // instance variables

  // Class (static) functions
  static void* classMethod1() ;
  static return_type classMethod2() ;
  static return_type classMethod3(param1_type parameter_varName) ;

  // Instance (member) functions
  return_type instanceMethod1(param1_type param1_varName,
                              param2_type param2_varName) ;

  return_type instanceMethod2WithParameter(param1_type param1_varName,
                                           param2_type param2_varName=default) ;
};
```

Паралел със C++

в Objective C пишем ... :

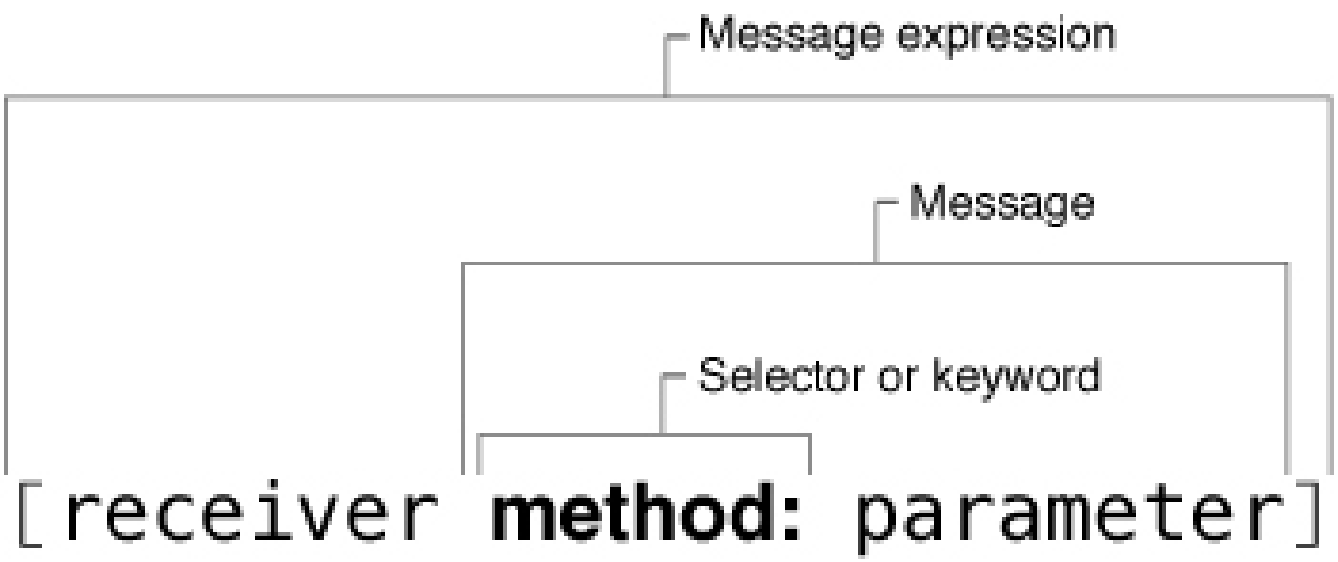
```
@interface classname : superclassname {
  // instance variables
}

+classMethod1;
+(return_type)classMethod2;
+(return_type)classMethod3:(param1_type)param1_varName;

-(return_type)instanceMethod1:(param1_type)param1_varName :
(param2_type)param2_varName;

-(return_type)instanceMethod2WithParameter:(param1_type)param1_varName
andOtherParameter:(param2_type)param2_varName;

@end
```

Код / съдържание / (.m/.h)

- За разлика от C++, заглавния (.h) файл и тялото (.m) на методите трябва да са разделени
- В заглавния файл @interface маркира клас дефиниции
- в (.m) файла имплементацията е маркирана от @implementation
- даден метод може да се отнася до целия клас (static) или до инстанция според префикса :
  - +(static)
  - -(instance)

Съобщения / терминология

! вместо да "извиква методи на обекти", Objective C изпраща "съобщения към обекти". за това и разширението (.m) : m = messages



Видимост

- По подразбиране всички променливи в един клас са @protected.

```
@interface Access: NSObject {
@public
int publicVarA;
int publicVarB;
@private
int privateVar;
@protected
int protectedVar;
}
@end
```

Класови променливи

- Клас, който държи броя на съществуващите обекти:
  - CountingInstances.h

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

static int count;

@interface ClassA: NSObject
+(int) getInstancesCount;
+(void) initialize;
@end
```

Класови променливи

- CountingInstances.m

```
#import " CountingInstances.h"
+(void) initialize {
count = 0;
}

@implementation ClassA
-(id) init {
self = [super init];
count++;
return self;
}
```

```
+(int) initCount {
return count;
}
@end

+(int) getInstancesCount{
return count;
}

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

Пример - Singleton pattern

```
@interface MySingleton : NSObject {
}
+ (MySingleton *) sharedInstance;
@end

@implementation MySingleton
+ (MySingleton *) sharedInstance {
static MySingleton *sharedSingleton;
@synchronized(self) {
if (!sharedSingleton) {
sharedSingleton = [[MySingleton alloc] init];
}
return sharedSingleton;
}
}
@end
```

Наследяване, протоколи

- Всеки клас наслдява точно един базов, но може да отговаря на много протоколи
  - self и super
  - протоколи – декларации на методи, който обект трябва да дефинира
    - @optional и @required (по подразбиране)

```
@protocol Locking
- (void)lock;
- (void)unlock;
@end
@protocol Printing
- (void)print; // may include only method declarations here;
@end
```

Наследяване, протоколи

```
@interface SubClass : BaseClass <Locking, Printing>
// nothing here
@end
```

- Важно: Използвайте Reflection за да сте сигурни, че optional метод е имплементиран.
  - if([recipient respondsToSelector: sel]) {...}

Наследяване, протоколи

- Променливи

```
// in the class interface:
- (void) methodName: (id<Locking>) obj;

// in some class method:
id<Printing> varA = [[SubClass alloc] init];
[varA print];

id<Locking> varB = [[SubClass alloc] init];
[self methodName: varB];
```

NSObject

- Почти всички класове в Cocoa наследяват NSObject
  - description – стрингово представяне на обекта, най-често използвано за debug

    ```
    NSArray * myArray = [NSArray arrayWithObjects: @“string1", @“string2", @“bla", nil];
    NSLog(@”MyArray: %@”, myArray);
    ```
  - isKindOfClass:, isMemberOfClass:, conformsToProtocol:
  - respondsToSelector:, methodForSelector:
  - performSelector:withObject:

Категории

- Алтернатива на подкласовете
  - позволяват добавяне на методи към вече съществуващи класове
  - позволяват променяне на поведението на съществуващи класове (override)

```
@interface ClassToModify (category)
// methods go here
@end
```

- Важно: Когато правите промени в клас чрез категория, променяте поведението на всички негови инстанции в приложението…

Пример:

NSString+Reverse.h

```
@interface NSString (Reverse)
-(NSString *) reverseString;
@end
```

NSString+Reverse.m

```
@implementation NSString (Reverse)
-(NSString *) reverseString {
// Some code to reverse the string
}
@end
```

main.m:

```
#import <Foundation/Foundation.h>
#import "NSString+Reverse.h"
int main (int argc, const char * argv[]) {
NSString *str =
[NSString alloc] initWithString: @“Bla"];
NSString *rev = [str reverseString];
// use the reversed string

return 0;
}
```

LOG
O

Благодаря за вниманието!

Въпроси?