



Politechnika Wroclawska

Programowanie w języku  
**Objective C**

(wprowadzenie)

Warsztaty Programowania Urządzeń Mobilnych

dr Marek Piasecki



# Najważniejsze symbole / operatory

<b>*.h   *.m</b>	<b>H</b> header file, <b>M</b> ethod / <b>iM</b> plementation file
<b>[ ]</b>	Wywołanie metody obiektu → wysłanie komunikatu np. <b>[gun fire];</b> (C/C++) → gun->fire();
<b>@</b>	Definiowanie elementów Objective C np. Stała tekstowa NSString: <b>@”Dzień dobry”</b>
<b>NS.....</b>	<b>N</b> eXT <b>S</b> tep OS
<b>NSObject</b>	Klasa bazowa wszystkich klas NS.....      ( Java → Object )
<b>id</b>	Wskaźnik na obiekt klasy NS.....      ( C/C++ → void* )
<b>nil</b> <b>NULL</b>	(id) 0 ← identyfikator nieistniejącego obiektu (void*) 0



# Najważniejsze symbole / operatory

<b>#import "....."</b> <b>#import &lt;.....&gt;</b>	~ <b>#include "....."</b> ~ <b>#include &lt;.....&gt;</b>
<b>BOOL, YES, NO</b>	~ <b>bool, true, false</b>
<b>NSLog(@"....", ... );</b>	~ <b>printf( "...", ... );</b>
<b>alloc</b> <b>dealloc</b>	- metoda alokująca pamięć (pierwsza faza konstrukcji) - destruktor
<b>init, initWith...</b>	metoda inicjalizująca pamięć (druga faza konstrukcji)
<b>retain</b> <b>release</b>	powiększa licznik referencji ( retainCount ++ ) pomniejsza licznik referencji ( retainCount --)
<b>strong</b> <b>weak</b>	- właściciel obiektu (blokuje zwolnienie obiektu przez innych) - użytkownik obiektu (nie blokuje, od iOS5 automatycznie zerowany)



# „Hello world !”

- hello.m

```
#import <stdio.h>

int main( int argc, const char *argv[] ) {
    printf( "hello world\n" );
    return 0;
}
```

- output

```
hello world
```



# Definiowanie interfejsu nowej klasy

## ■ Fraction.h

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

@interface Fraction: NSObject {
    int numerator;
    int denominator;
}

-(void) print;
-(void) setNumerator: (int) n;
-(void) setDenominator: (int) d;
-(int) numerator;
-(int) denominator;
@end
```



# Definiowanie implementacji nowej klasy

## ■ Fraction.m

```
#import "Fraction.h"
#import <stdio.h>

@implementation Fraction
-(void) print {
    printf( "%i/%i", numerator, denominator );
}

-(void) setNumerator: (int) n {
    numerator = n;
}

-(void) setDenominator: (int) d {
    denominator = d;
}

-(int) denominator {
    return denominator;
}

-(int) numerator {
    return numerator;
}
@end
```



# Wykorzystanie nowej klasy Fraction

## ■ main.m

```
#import <stdio.h>
#import "Fraction.h"

int main( int argc, const char *argv[] ) {
    // create a new instance
    Fraction *frac = [[Fraction alloc] init];

    // set the values
    [frac setNumerator: 1];
    [frac setDenominator: 3];

    // print it
    printf( "The fraction is: " );
    [frac print];
    printf( "\n" );

    // free memory
    [frac release];

    return 0;
}
```



# Metody z wieloma parametrami

- Fraction.h

```
...  
-(void) setNumerator: (int) n andDenominator: (int) d;  
...
```

- Fraction.m

```
...  
-(void) setNumerator: (int) n andDenominator: (int) d {  
    numerator = n;  
    denominator = d;  
}  
...
```





# Wykorzystanie metody wieloparametrowej

## ■ main.m

```
#import <stdio.h>
#import "Fraction.h"

int main( int argc, const char *argv[] ) {
    // create a new instance
    Fraction *frac = [[Fraction alloc] init];
    Fraction *frac2 = [[Fraction alloc] init];

    // set the values
    [frac setNumerator: 1];
    [frac setDenominator: 3];

    // combined set
    [frac2 setNumerator: 1 andDenominator: 5];

    // print it
    printf( "The fraction is: " );
    [frac print];
    printf( "\n" );

    // print it
    printf( "Fraction 2 is: " );
    [frac2 print];
    printf( "\n" );

    // free memory
    [frac release];
    [frac2 release];

    return 0;
}
```



# Konstruktory ( init, initWith.. )

- Fraction.h

```
...  
-(Fraction*) initWithNumerator: (int) n denominator: (int) d;  
...
```

- Fraction.m

```
...  
-(Fraction*) initWithNumerator: (int) n denominator: (int) d {  
    self = [super init];  
  
    if ( self ) {  
        [self setNumerator: n andDenominator: d];  
    }  
  
    return self;  
}  
...
```



# Wykorzystanie konstruktorów

## ■ main.m

```
#import <stdio.h>
#import "Fraction.h"

int main( int argc, const char *argv[] ) {
    // create a new instance
    Fraction *frac = [[Fraction alloc] init];
    Fraction *frac2 = [[Fraction alloc] init];
    Fraction *frac3 = [[Fraction alloc] initWithNumerator: 3 denominator: 10];

    // set the values
    [frac setNumerator: 1];
    [frac setDenominator: 3];

    // combined set
    [frac2 setNumerator: 1 andDenominator: 5];

    // print it
    printf( "The fraction is: " );
    [frac print];
    printf( "\n" );

    printf( "Fraction 2 is: " );
    [frac2 print];
    printf( "\n" );

    printf( "Fraction 3 is: " );
    [frac3 print];
    printf( "\n" );

    // free memory
    [frac release];
    [frac2 release];
    [frac3 release];

    return 0;
}
```



# Modyfikatory widoczności składowych

Modyfikator	Zasięg
<b>@private</b>	atrybut widoczny jedynie dla metod klasy, która go zadeklarowała
<b>@protected</b>	Atrybut widoczny dla metod klasy, która go zadeklarowała, oraz dla klas potomnych (dziedziczących)
<b>@public</b>	atrybut publiczny, widoczny bez ograniczeń
<b>@package</b>	zasięg pakietowy, widoczny w pakiecie/module/pliku w którym jest zdefiniowana implementacja klasy



# Modyfikatory widoczności (2)

- Access.h

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

@interface Access: NSObject {
    @public
        int publicVar;
    @private
        int privateVar;
        int privateVar2;
    @protected
        int protectedVar;
}
@end
```

- Access.m

```
#import "Access.h"

@implementation Access
@end
```



# Testowanie widoczności zmiennych

- main.m

```
#import "Access.h"
#import <stdio.h>

int main( int argc, const char *argv[] ) {
    Access a = [[Access alloc] init];

    // works
    a->publicVar = 5;
    printf( "public var: %i\n", a->publicVar );

    // doesn't compile
    //a->privateVar = 10;
    //printf( "private var: %i\n", a->privateVar );

    [a release];
    return 0;
}
```



# Metody instancji ↔ Metody klasy

- ClassA.h

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

static int count;

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

- ClassA.m

```
#import "ClassA.h"

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

+(int) initCount {
    return count;
}

+(void) initialize {
    count = 0;
}
@end
```



# Wykorzystanie metod klasy ClassA

- main.m

```
#import "ClassA.h"
#import <stdio.h>

int main( int argc, const char *argv[] ) {
    ClassA *c1 = [[ClassA alloc] init];
    ClassA *c2 = [[ClassA alloc] init];

    // print count
    printf( "ClassA count: %i\n", [ClassA initCount] );

    ClassA *c3 = [[ClassA alloc] init];

    // print count again
    printf( "ClassA count: %i\n", [ClassA initCount] );

    [c1 release];
    [c2 release];
    [c3 release];

    return 0;
}
```

- output

```
ClassA count: 2
ClassA count: 3
```





# Dziedziczenie: Rectangle → NSObject

## ■ Rectangle.h

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

@interface Rectangle: NSObject {
    int width;
    int height;
}

-(Rectangle*) initWithWidth: (int) w height: (int) h;
-(void) setWidth: (int) w;
-(void) setHeight: (int) h;
-(void) setWidth: (int) w height: (int) h;
-(int) width;
-(int) height;
-(void) print;
@end
```



# Dziedziczenie: Rectangle → NSObject

- Rectangle.m

```
#import "Rectangle.h"
#import <stdio.h>

@implementation Rectangle
-(Rectangle*) initWithWidth: (int) w height: (int) h {
    self = [super init];

    if ( self ) {
        [self setWidth: w height: h];
    }

    return self;
}

-(void) setWidth: (int) w {
    width = w;
}

-(void) setHeight: (int) h {
    height = h;
}
```



# Dziedziczenie: Square → Rectangle →

- Square.h

```
#import "Rectangle.h"

@interface Square: Rectangle
-(Square*) initWithSize: (int) s;
-(void) setSize: (int) s;
-(int) size;
@end
```

- Square.m

```
#import "Square.h"

@implementation Square
-(Square*) initWithSize: (int) s {
    self = [super init];

    if ( self ) {
        [self setSize: s];
    }

    return self;
}

-(void) setSize: (int) s {
    width = s;
    height = s;
}
```



# Typy dynamiczne (introspekcja)

(BOOL) <u>isKindOfClass</u> : classObj	is object a descendent or member of classObj
-(BOOL) <u>isMemberOfClass</u> : classObj	is object a member of classObj
-(BOOL) <u>respondsToSelector</u> : selector	does the object have a method named by the selector
+(BOOL) <u>instancesRespondToSelector</u> : selector	does an object created by this class have the ability to respond to the specified selector
-(id) <u>performSelector</u> : selector	invoke the specified selector on the object

```
#import "Square.h"
#import "Rectangle.h"
#import <stdio.h>

int main( int argc, const char *argv[] ) {
    Rectangle *rec = [[Rectangle alloc] initWithWidth: 10 height: 20];
    Square *sq = [[Square alloc] initWithSize: 15];

    // isMemberOfClass
    // true
    if ( [sq isMemberOfClass: [Square class]] == YES ) {
        printf( "square is a member of square class\n" );
    }
    // false
    if ( [sq isMemberOfClass: [Rectangle class]] == YES ) {
        printf( "square is a member of rectangle class\n" );
    }
    // false
    if ( [sq isMemberOfClass: [NSObject class]] == YES ) {
        printf( "square is a member of object class\n" );
    }
    // isKindOfClass
    // true
    if ( [sq isKindOfClass: [Square class]] == YES ) {
        printf( "square is a kind of square class\n" );
    }
    // true
    if ( [sq isKindOfClass: [Rectangle class]] == YES ) {
        printf( "square is a kind of rectangle class\n" );
    }
}
```



# Fundation Framework Classes

- **NSObject**
- **NSString**, NSMutableString
- **NSArray**, NSMutableArray
- **NSDictionary**, NSMutableDictionary
- **NSSet**, NSMutableSet, NSOrderedSet
- **NSNumber**
- **NSValue**
- **NSData**