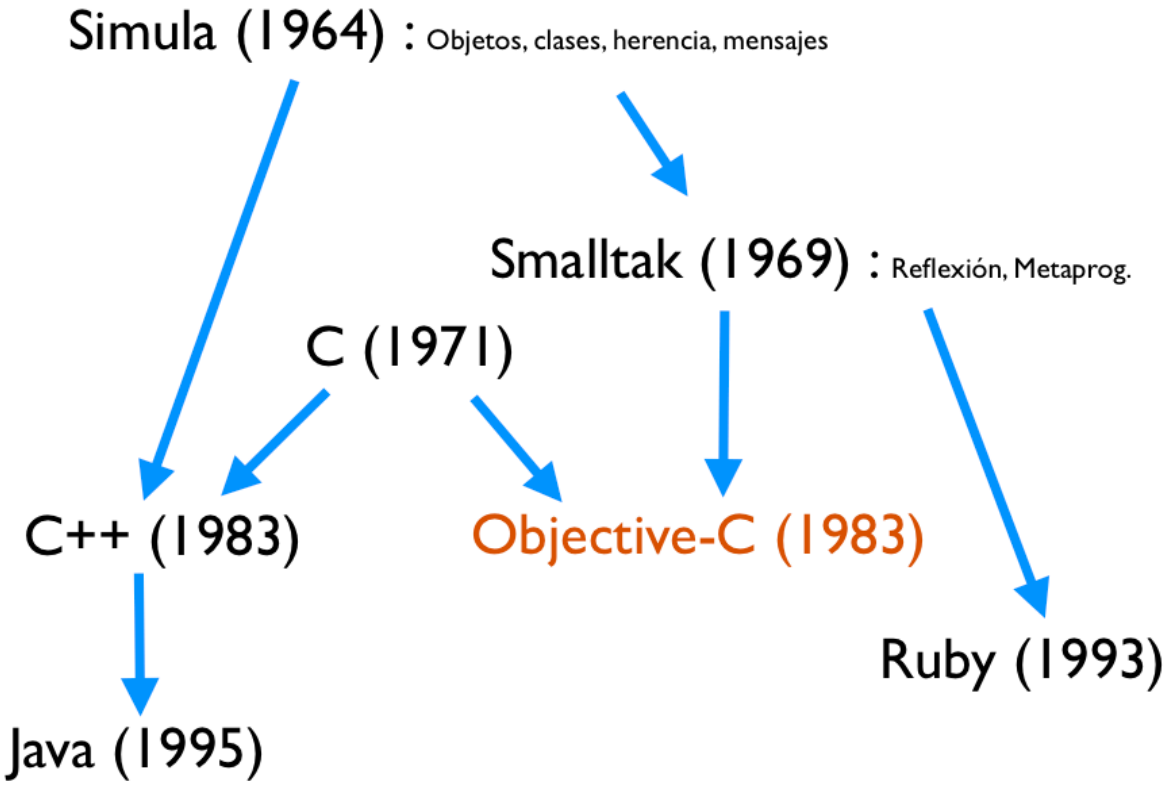
Introducción a Objective-C

Hola!

Consultor en adaptación de tecnologías móviles
Ingeniero especalista en iOS y Android
Twitter: @omargomez
LinkedIn: http://co.linkedin.com/in/themobileworld/
Correo: omargomez@gmail.com

Árbol genealógico de Objetive-C



El lenguaje de programación:
Objective-C

(The History of Programming Languages: http://fwd4.me/QS8)

Llamando métodos sin parámetros

Java	int l = str.length();
Objective-c	int l = [str length];

Diferencias:

- El llamado de las operaciones se debe hacer entre corchetes ‘[]’
- No se usan paréntesis para agrupar los parámtros
- No se usa el punto. Se separa el nombre del método con uno o más espacios

Llamando métodos con un parámetro

Java	`String sub = str.substring(1);`
Objective-c	`NSString *sub = [str substringFromIndex:1];`

Diferencias:

- Se usan los dos puntos ‘:’ para separar el nombre del método del parámetro

Llamando métodos con dos o más parámetros

Java	`String new = str.replace('a', 'A');`
Objective-c	`NSString *new = [str stringByReplacingOccurrencesOfString:@"a" withString:@"A"];`

Diferencias:

- El segundo y los siguientes parámetros deben separarse de los anteriores mediante una cadena con la estructura "indentificador:". La idea del identificador es que ayude a que el código sea más legible

- Nótese que cuando existe un solo parámetro, el nombre del método es a su vez el prefijo del primer parámetro

Firma de métodos

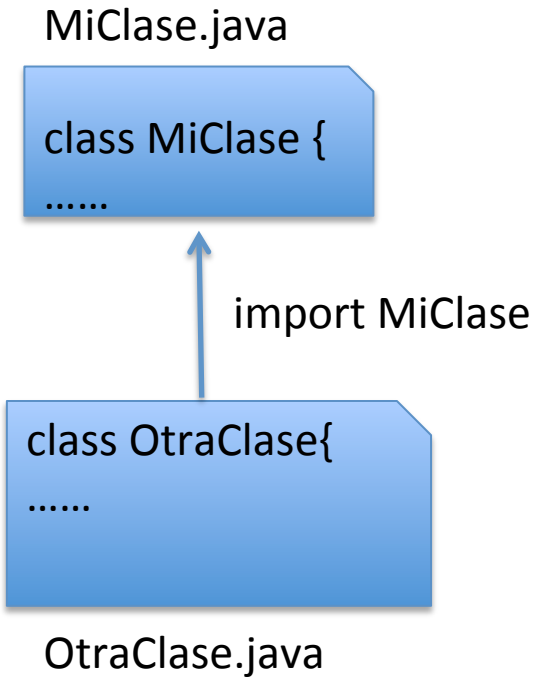
En java la firma de un método es el conjunto formado por su nombre y los tipos de sus parámetros, por eso pueden existir dos métodos con nombres iguales pero tipos de parámetros diferentes:

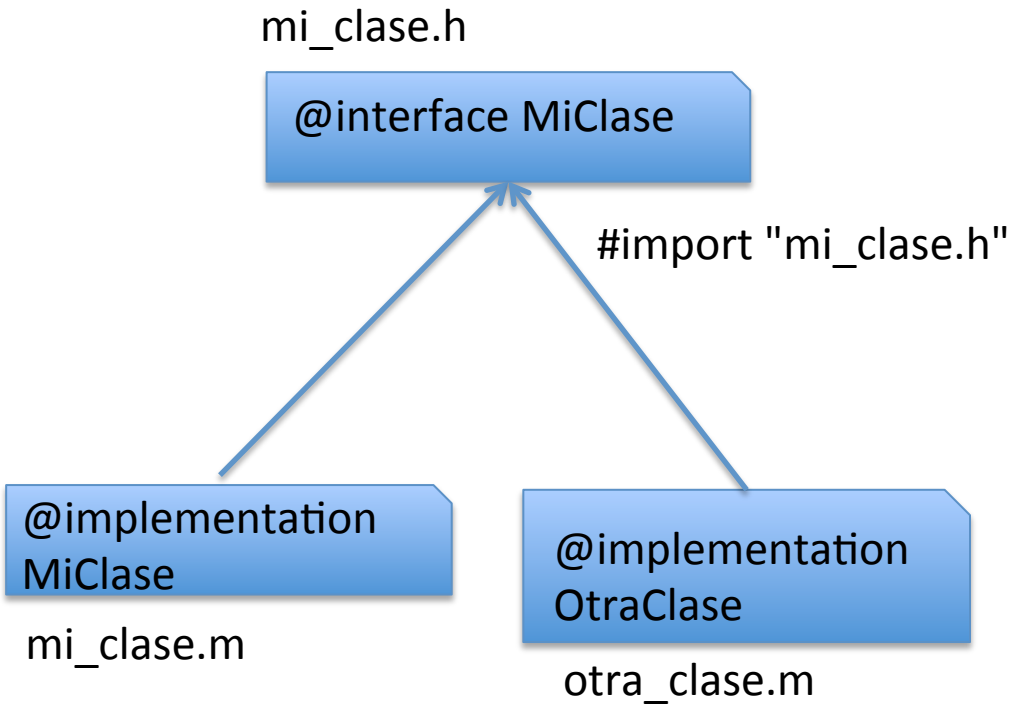
- indexOf(int ch) -> La firma es *('indexOf', int)*
- indexOf(int ch, int fromIndex) -> la firma es *('indexOf', int, int)*

En Objective-C la firma del método es el conjunto formado por los nombres del métodos y los identificadores de los parámetros:

- - (NSRange)rangeOfString:(NSString *)aString -> La firma es "*rangeOfString:*"
- - (NSRange)rangeOfString:(NSString *)aString options: (NSStringCompareOptions)mask -> La firma es "*rangeOfString:options:*"

Organización de fuentes





Java ***Objective-C***

Ejercicio: Ej 00 uso_metodos

Declaración de clases (Java)

```java
public class Fraccion extends Object {

    int numerador;
    int denominador;

    void sumar( Fraccion otra ) {
        // definicion
    }

    void multiplicar( Fraccion otra ) {
        // definicion
    }

}
```

Declaración de clases (Objective-C)

```
// Declaración de interface (fraccion.h)
@interface Fraccion : NSObject
{
    // Datos de instancia (Instance data - ivars)
    int numerador;
    int denominador;
}

// Métodos
-(void) sumar:(Fraccion *) otra;
-(void) multiplicar:(Fraccion *) otra;

@end

// Definición de interface (fraccion.m)
@implementation Fraccion

-(void) sumar:(Fraccion *) otra
{
    // definicion de la suma
}

-(void) multiplicar:(Fraccion *) otra
{
    // definicion de la suma
}

@end
```

Declaración de clases (1/2)

- La declaración de la clase y su definición deben ir en archivos diferentes (*.h y *.m respectivamente).
- La declaración de la clase cumple con la siguiente plantilla:

```
@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
```

- Con respecto a los tipos:
  - Son los mismos tipos usados en C (int, float, … http://goo.gl/hp8DC)
  - typedef a tipos de C (typedef long NSInteger;)
  - Apuntadores a clases (NSString *)
  - Apuntadores instancias sin tipo (id)

Declaración de clases (2/2)

- La definición de la clase cumple con la plantilla:

```
@implementation classname
+ (return_type)classMethod {
    // implementation
}
- (return_type)instanceMethod {
    // implementation
}
@end
```

Creación de instancias

Java

```
String  nuevo = new String("Hola Mundo");
```

Objective-c

```
NSString *s = [[NSString alloc] initWithString:@"Hola Mundo"]; // Contador de referencias en 1
NSString *s = [NSString stringWithString:@"Hola Mundo"]; // Lo manda al autorelease pool
```

Notas:

- alloc: Asigna la memoria a la nueva instancia
- init* : Los métodos que empieza con 'init' inician la memoria recien conseguida
- autorelease pool: area donde los objetos creados viven temporalmente según que tan frecuentemente el pool se limpie.

Modificadores de acceso

```
/* Modificadores de acceso */

@interface ClaseXYZ : NSObject
{
@private
    int x;

@protected
    int y;

@public
    int z;
}

-(int) getX;
-(int) getY;
-(int) getZ;

@end

@implementation ClaseXYZ

// Todos los métodos son
// publicos
-(int) getX { return x; }
-(int) getY { return y; }
-(int) getZ { return z; }

@end
```

Modificadores de acceso

- Los modificadores de acceso (@private, @protected, @public) se aplican sólo a los atributos de instancia. Todos métodos declarados son públicos
- Métodos privados (manejados con categoría)

Métodos inicializadores

```objc
/* Inicializadores */

/* Los inicialzadores deben llamarse tan pronto
 * la memoria para la clase ha sido reservada (alloc)
 */
@interface Circulo : NSObject
{
    float radio;
    float centroX, centroY;
}
-(id) init; // 1) El nombre debe empezar por 'init'
-(id) initWithRadio:(float) r;
-(id) initWithRadio:(float)r andX:(float)x andY:(float)y;
@end

@implementation Circulo
-(id) init
{
    // LLama al inicializador de la clase padre
    if (!(self = [super init]))
        return nil; // falla si el padre falla

    radio = 1.0;
    centroX = centroY = 0;
    // si la inicializacion falla debe retornar 'nil'
    return self; // Retorna la referencia del objeto
}

-(id) initWithRadio:(float) r
{
    if (!(self = [super init]))
        return nil;

    radio = r;
    centroX = centroY = 0;
    return self;
}

-(id) initWithRadio:(float)r andX:(float)x andY:(float)y
{
    if (!(self = [super init]))
        return nil;

    radio = r;
    centroX = x;
    centroY = y;
    return self;
}
@end
```

Métodos

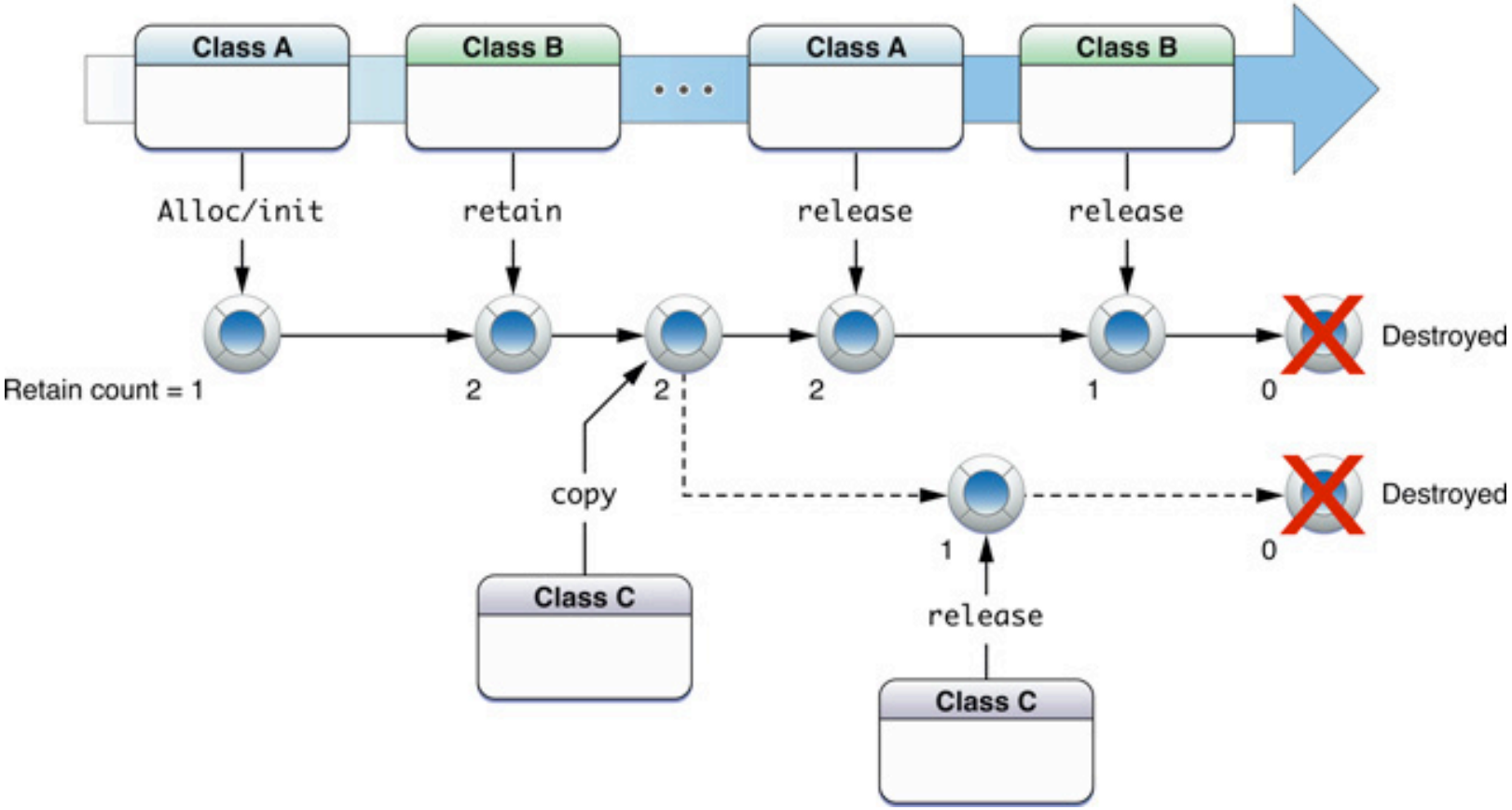
```
/* Métodos */

@interface Circulo : NSObject
{
    float radio;
    float centroX, centroY;
}

// 1) '-' Se como prefilo de los métodos de instancia
//    '+' Se usa como prefijo de los métods de clase
-(float) getArea;
+(float) getPI;

// Tanto como el tipo de retorno, como los parámetros
// se encierran entre paréntesis
//Los parámetros se separan por ':'
-(void) trasladar:(float)x :(float)y;

//Comúnmente un label se asocia a cada parámetro
// (El label del primer parámetros es el nombre del
//  método). El uso de labels ayuda en la claridad
// del código
-(void) trasladarEnX:(float)x enY:(float)y;

// El nombre de un método puede ser igual al de
// un atributo
-(float) radio;

@end

@implementation Circulo

#define PI_VAL 3.141592

-(float) getArea { return PI_VAL*radio*radio; }
+(float) getPI { return PI_VAL; }
-(void) trasladar:(float)x :(float)y
{ centroX += x; centroY += y; }
-(void) trasladarEnX:(float)x enY:(float)y
{ centroX += x; centroY += y; }
-(float) radio { return radio; }

@end
```

Mensajes

```
void usoCirculo()
{
    // getPI es un método de clase.
    float p = [Circulo getPI];
    printf("%f\n", p);

    // Tres maneras de crear un circulo
    Circulo *c1 = [[Circulo alloc] init];
    Circulo *c2 = [[Circulo alloc] initWithRadio:10.0];
    Circulo *c3 = [[Circulo alloc] initWithRadio:20.0 andX:2.0 andY:1.0];

    printf("%f\n", [c1 getArea]);
    printf("%f\n", [c2 radio]);

    [c3 trasladar:2.0:3.0];
    [c3 trasladarEnX:4.0 enY:5.0];

    // Selectors (apuntadores a funciones)
    SEL s1 = @selector(radio);
    SEL s2 = @selector(trasladarEnX:enY:);

    // Llama el método 'radio' sobre la instancia
    // c1
    [c1 performSelector:s1];

    // Note el uso de los parámetros como objetos
    [c2 performSelector:s2
            withObject:[NSNumber numberWithFloat:1.0]
            withObject:[NSNumber numberWithFloat:2.0]
     ];

    [c1 release];
    [c2 release];
    [c3 release];
}
```

Ej 01 - Mi primera clase

Manejo de memoria



Escenario #1: Yo creo yo destruyo (alloc/init/ release)

```objc
@interface Punto : NSObject
{
    float x,y;
}

-(id) init;
-(id) initWithX:(float)_x andY:(float)_y;
@end

@implementation Punto
-(id) init
{
    if (!(self = [super init]))
        return nil;
    x = 0; y = 0;
    return self;
}

-(id) initWithX:(float)_x andY:(float)_y
{
    if (!(self = [super init]))
        return nil;
    x = _x; y = _y;
    return self;
}
@end
```

```objc
// Crea ...
Punto *p1 = [[Punto alloc] init];
Punto *p2 = [[Punto alloc] initWithX:1.0 andY:2.0];

// Hace algo con los puntos ....

// Libera...
[p1 release];
[p2 release];
```

Escenario #2: Yo creo, otro destruye (alloc/ init/autorelease)

```objc
@interface Punto : NSObject
{
    float x,y;
}

+(id) punto;
+(id) puntoWithX:(float)_x andY:(float)_y;
@end

@implementation Punto

+(id) punto
{
    id inst = [[[self class] alloc] init];

    return [inst autorelease];
}

+(id) puntoWithX:(float)_x andY:(float)_y;
{
    id inst = [[[self class] alloc] initWithX:_x andY:_y];

    return [inst autorelease];
}
@end
```

```objc
Punto *p1 = [Punto punto];
Punto *p2 = [Punto puntoWithX:1.0 andY:2.0];
Punto *p3 = [[[Punto alloc] init] autorelease];

// Hace algo con los puntos ...

// No es necesario liberarlos
```

Escenario #3: Composicion de instancias

```objc
@interface Figura : NSObject
{
    Punto *centro;
}

-(id) init;
-(void) dealloc;
// Accessors
-(Punto *) centro;
-(void) setCentro:(Punto *) otro;

@end
```

```objc
// Uso
Figura *f = [[Figura alloc] init];

[f setCentro:[Punto puntoWithX:1.0 andY:2.0]];
Punto *c = [f centro];

[f release];
```

```objc
-(id) init
{
    if (!([super init]))
        return nil;

    centro = [[Punto alloc] init];
    return self;
}

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

-(Punto *) centro
{
    return centro;
}

-(void) setCentro:(Punto *) otro;
{
    if (centro != otro) {
        [centro release];
        centro = [otro retain];
    }
}
```

Uso de propiedades

```objc
// Uso de propiedades
Rectangulo *r = [[Rectangulo alloc] init];

r.leftTop = [Punto punto];
r.rightBottom = [Punto puntoWithX:1.0 andY:2.0];

[r release];
```

```objc
@interface Rectangulo : NSObject
{
    Punto *leftTop;
    Punto *rightBottom;
}
@property (retain) Punto* leftTop;
@property (retain) Punto* rightBottom;

@end

@implementation Rectangulo

@synthesize leftTop;
@synthesize rightBottom;

-(void) dealloc
{
    [leftTop release];
    [rightBottom release];
    [super dealloc];
}

@end
```

Referencias

- Una buena “CheatSheet” (referencia rápida) del lenguaje: https://github.com/iwasrobbed/Objective-C-CheatSheet