Universidad de Salamanca

IHM - OBJECTIVE-C

Depto. Informática y Automática
Máster en Sistemas Inteligentes
Dr. J.R. García-Bermejo Giner

IHM - Objective-C

Segunda parte - Objective-C

Vamos a estudiar brevemente el lenguaje Objective-C

- Mecanismo de declaración de clases.
- Mecanismo de declaración de categorías.
- Mecanismo de declaración de protocolos.
- Otros mecanismos.

IHM - Objective-C

Primeros pasos en Objective-C

- Hola, Mundo! (manual)

- Hola, Mundo! (XCode)

El compilador de Objective-C está disponible en gnuc, pero la biblioteca de clases asociada debe instalarse por separado salvo en Mac OS X.

Una dirección curiosa: http://blog.lyxite.com/2008/01/compile-objective-c-programs-using-gcc.html

IHM - Objective-C

Notas
=====

Los signos diacríticos y en general los códigos ASCII extendidos requieren acuerdo entre el formato del código fuente y la terminal empleada. (DEMO)

Objective-C conoce y utiliza UTF8.

- Hola, Mundo! (manual)

```
gcc -L /System/Library/Frameworks/Foundation.framework/Foundation *.m
```

- Hola, Mundo! (XCode) (ver)

El compilador de Objective-C está disponible en gnuc, pero la biblioteca de clases asociada debe instalarse por separado salvo en Mac OS X.

Una dirección curiosa: http://blog.lyxite.com/2008/01/compile-objective-c-programs-using-gcc.html

IHM - Objective-C

Un programa completo en Objective-C

Archivo: ListaDeAlimentos.h

```
#import <Cocoa/Cocoa.h>
#import <AppKit/NSGraphics.h>

@interface ListaDeAlimentos : NSObject {
    NSMutableDictionary * lista;
}

@property(readonly) NSMutableDictionary * lista;
-(id)init;
-(float)caloriasPorCienGramosDe:(NSString*)alimento;
-(void)registrar:(NSString*)alimento caloriasPorCienGramos:(float)valor;
-(void)eliminar:(NSString*)alimento;

@end
```

IHM - Objective-C

Un programa completo en Objective-C

Archivo: ListaDeAlimentos.m

```
#import "ListaDeAlimentos.h"
@implementation ListaDeAlimentos
@synthesize lista;
-(id)init {
    [super init];
    lista = [[NSMutableDictionary alloc] init];
    return self;
}
-(float)caloriasPorCienGramosDe:(NSString*)alimento {
    NSString * calorias = [lista objectForKey:alimento];
    return calorias ? [calorias floatValue] : -1.0;
}
-(void)registrar:(NSString*)alimento caloriasPorCienGramos:(float)valor {
    NSString * copia = [alimento copy];
    [lista setObject:[NSNumber numberWithFloat:valor] forKey:copia];
}
-(void)eliminar:(NSString*)alimento {
    if(alimento)
        [lista removeObjectForKey:alimento];
    else
        NSBeep();
}
@end
```

IHM - Objective-C

Un programa completo en Objective-C

Archivo: Calorimetro.m

```
#import <Foundation/Foundation.h>
#import <AppKit/NSGraphics.h>
#import "ListaDeAlimentos.h"

int main (int argc, const char * argv[]) {
    ListaDeAlimentos * p = [[ListaDeAlimentos alloc] init];
    NSMutableDictionary * listaAlmacenada = p.lista;

    [p registrar:@"Fresas" caloriasPorCienGramos:35.0];
    [p registrar:@"Moras" caloriasPorCienGramos:59.0];


    for(id alimento in listaAlmacenada)
    {
        printf("Alimento: %-10s ---> Calorías: %5.1f\n", [alimento UTF8String],
                            [[listaAlmacenada objectForKey:alimento] floatValue]);
    }

    return 0;
}
```

IHM - Objective-C

Un programa completo en Objective-C - Resultado de la ejecución de este programa

```
[Session started at 2008-06-07 19:18:26 +0200.]
Alimento: Fresas      ---> Calorías:  35.0
Alimento: Moras       ---> Calorías:  59.0

The Debugger has exited with status 0.
```

IHM - Objective-C

Comentarios
==========

- Se ha utilizado un ejemplar de NSMutableDictionary como estructura de datos para almacenar información.

- Se ha empleado la sintaxis de propiedades para acceder a lista.

- Se ha enviado el mensaje registrar:caloriasPorCienGramos: a un ejemplar de ListaDeAlimentos.

- Se ha empleado la sintaxis de enumeración rápida (iteradores) para recorre la lista de parejas plato/calorías.

- Se han efectuado conversiones a formatos imprimibles.

IHM - Objective-C

Declaración de clases en Objective-C

Las clases constan de interfaz e implementación; normalmente se escribe la interfaz en un archivo de encabezado (.h) y la implementación en un archivo de implementación (.m). El contenido de un archivo de interfaz (NombreDeLaClase.h) es como sigue:

```
@interface NombreDeLaClase : NombreDeLaSuperclase <NombreProtocolo_01,...> {

// class NombreDeLaClase extends NombreDeLaSuperclase implements NombreProtocolo...

    // Zona de declaración de variables de ejemplar (no hay variables de clase!)
    -(tipo_1) variableDeEjemplar_1;
    ...
    -(tipo_M) variableDeEjemplar_N;
}

    // Zona de declaración de métodos
    +(tipo) metodoDeclase;
    ...
    -(tipo) metodoDeEjemplar:(tipo)argumento etiqueta:(tipo)otroArgumento;


@end
```

IHM - Objective-C

Declaración de clases en Objective-C
====================================
Archivo: NombreDeLaClase.h

En la zona de declaración de variables de ejemplar se admite la presencia de cuatro indicadores de ámbito:

@private   - visible en métodos de la clase únicamente.
@protected - visible en métodos de la clase y de descendientes directos (por defecto).
@public   - visible en cualquier método de cualquier clase (no se recomienda).
@package   - en máquinas de 64 bits, equivale a @public dentro de la misma imagen
              en que se implementa la clase, pero fuera de esa imagen es como @private.

Por omisión, todas las variables de ejemplar se encuentran en la zona @protected

```
@interface NombreDeLaClase : NSObject {
    -(tipo_1) variableDeEjemplar_1; // protegida
    @private
    -(tipo_2) variableDeEjemplar_2; // privada
    @protected
    -(tipo_3) variableDeEjemplar_3; // protegida
    @package
    -(tipo_4) variableDeEjemplar_4; // de paquete
    @public
    -(tipo_M) variableDeEjemplar_N; // pública, no se recomienda
}

    // Zona de declaración de métodos
    …
```

IHM - Objective-C

Declaración de clases en Objective-C
==========

Archivo: NombreDeLaClase.h

Tras la llave que marca el fin de la zona de declaración de variables se halla la zona de declaración de métodos. Hay dos tipos de métodos:

```
-- de clase

+(id) alloc;
+(id) init;
+(id) initWithValue:(id)someValue;

-- de ejemplar

-(void)setVariable:(tipo)valor;
-(float)sumaDeValores;
-(NSMutableArray*)contenido:(id)obj desdeAqui:(int)desde hastaAqui:(int)hasta;
```

Los métodos de clase (init, alloc) sirven para reservar memoria para nuevos ejemplares y para dar valores iniciales a esos ejemplares. Los métodos de ejemplar actúan sobre las variables de ejemplar, son los métodos habituales de una clase.

IHM - Objective-C

Declaración de clases en Objective-C
====================================
Archivo: NombreDeLaClase.h

La forma de declarar el tipo proporcionado por un método, y los argumentos que recibe es peculiar. El tipo proporcionado se coloca a la derecha del signo +/- (método de clase o de ejemplar), y va entre paréntesis. Los argumentos no van entre paréntesis a la derecha del nombre del método. Se declaran en la forma

```
etiqueta : (tipo) nombreDeArgumento
```

El nombre del método hace las veces de etiqueta del primer argumento. La etiqueta se utiliza después en la llamada al método, permitiendo al usuario recordar fácilmente el significado del argumento. Véanse algunos ejemplos de declaración de métodos.

```
-(float)sumaDeValores;
-(void)setVariable:(tipo)valor;
-(NSMutableArray*)contenido:(id)obj desdeAqui:(int)desde hastaAqui:(int)hasta;
```

Estos métodos se invocarían en la forma siguiente:

```
suma = [objeto sumaDeValores];
otroObjeto.Variable=33;
NSMutableArray * c = [unObjeto contenido:otroObjeto desdeAqui:33 hastaAqui:44];
```

IHM - Objective-C

Declaración de clases en Objective-C
==========

Archivo: NombreDeLaClase.m

El archivo de implementación importa al de interfaz, salvo que residan en un mismo fichero. A continuación se especifica la implementación de los métodos declarados.

```
#import “NombreDeLaClase.h”

@implementation NombreDeLaClase // No es preciso repetir superclase y protocolos
    +(id) init {
        [super init];
        ...
        return self;
    }
    ...
    -(tipo)nombreDeMetodo {
        ...
    }

    ...
    -(otroTipo)otroMetodo:(tipo)argumento conOtroParametro:(tipo)valor {
        ...
    }
@end
```

Los métodos ven todas las variables de ejemplar. Se cuenta con los identificadores automáticos self (como this) y super (que denota la clase precedesora).

IHM - Objective-C

El mecanismo de herencia

Objective-C ofrece un mecanismo de herencia simple. Las variables y métodos pertenecientes a la clase predecesora también son visibles en las clases derivadas, en las derivadas de éstas (“nietas”), etc. El especificador de ámbito @protected limita la visibilidad de esa manera precisamente.

La sintaxis es como sigue:

```
@interface NombreDeClaseDerivada : NombreDeClasePredecesora {

}

@end
```

Si se envía un mensaje a una clase y no puede responder a él por no estar implementado ese método en esa clase, se asciende por la jerarquía de herencia hasta hallar y ejecutar ese método, o hasta llegar a NSObject.

Las clases derivadas pueden añadir nuevas variables y métodos.

En caso de colisión, se puede acceder a las variables y métodos de la clase predecesora empleando la palabra reservada super.

Las clases derivadas pueden redefinir métodos de las clases predecesoras. Si se envía un mensaje a la clase derivada, se ejecutará el método definido en esa clase derivada.

IHM - Objective-C

El mecanismo de herencia

Los punteros de las clases predecesoras y de las derivadas son compatibles. Pero el punto de entrada para buscar métodos es siempre el real, es decir, el conjunto de métodos definidos en el puntero de que se disponga.

Si se asigna a un puntero de clase predecesora el puntero de un ejemplar de clase derivada y se invoca un método, se ejecutará el de la clase derivada (el método correspondiente al ejemplar de que se dispone realmente).

Si se asigna a un puntero de clase derivada el puntero de un ejemplar de clase predecesora y se invoca un método, pueden pasar dos cosas. Si el método está definido en la clase predecesora, se ejecuta el método de la clase predecesora. Si el método no está definido en la clase predecesora, se produce una excepción porque se ha invocado un método inexistente en la clase de cuyo puntero se dispone.

En resumen, al margen de la compatibilidad entre punteros de clases predecesoras y derivadas, el punto de entrada al código es el señalado por el puntero.

IHM - Objective-C

El mecanismo de herencia

Al mandar un mensaje, se consulta la tabla de métodos disponibles en el objeto señalado por el puntero, sea cual fuere el tipo del puntero en que está almacenada esa dirección.

Si el objeto es de este tipo	Se dispone de estos métodos	
	Heredados	Propios
Grossvater		init
		showVariables
Vater	init	showVariables
Sohn	init	showVariables
		metodoNuevo

Sólo se puede enviar el mensaje metodoNuevo a ejemplaresde Sohn.
El mensaje showVariables se puede enviar a ejemplares de Grossvater, Vater y Sohn. Cada uno responderá con su propia versión. Sólo se ejecutaría un método de una clase predecesora si en el ejemplar considerado no existiera ese método.

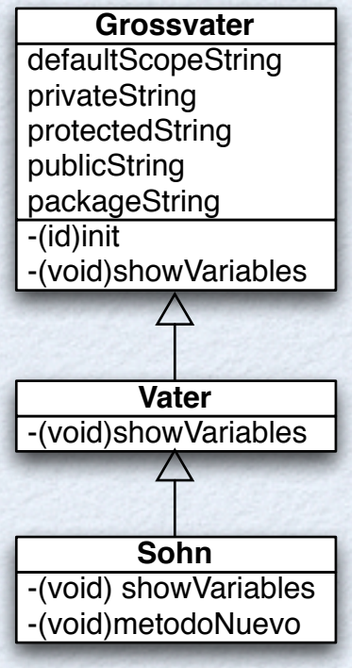
Es buena práctica de programación emplear como tipo estático el puntero de la clase base de la jerarquía. De ese modo se puede almacenar en ese puntero cualquier puntero de un ejemplar de las clases derivadas. Los mensajes válidos para la clase base lo serán para las clases derivadas. Esto es la base del concepto de protocolo (q. v.).

IHM - Objective-C

El mecanismo de herencia

Al mandar un mensaje, se consulta la tabla de métodos disponibles en el objeto señalado por el puntero, sea cual fuere el tipo del puntero en que está almacenada esa dirección.

Si el objeto es de este tipo	Se dispone de estos métodos	
	Heredados	Propios
Grossvater		init
		showVariables
Vater	init	showVariables
Sohn	init	showVariables
		metodoNuevo



Sólo se puede enviar el mensaje metodoNuevo a ejemplaresde Sohn.
El mensaje showVariables se puede enviar a ejemplares de Grossvater, Vater y Sohn. Cada uno responderá con su propia versión. Sólo se ejecutaría un método de una clase predecesora si en el ejemplar considerado no existiera ese método.

Es buena práctica de programación emplear como tipo estático el puntero de la clase base de la jerarquía. De ese modo se puede almacenar en ese puntero cualquier puntero de un ejemplar de las clases derivadas. Los mensajes válidos para la clase base lo serán para las clases derivadas. Esto es la base del concepto de protocolo (q. v.).

IHM - Objective-C

El mecanismo de herencia
==========

Archivo: Grossvater.h

```
//
//  Großvater.h
//  Ambitos

@interface Grossvater : NSObject {
    NSString * defaultScopeString; // Protected
@private
    NSString * privateString;
@protected
    NSString * protectedString;
@public
    NSString * publicString;
    @package
    NSString * packageString;
}


-(id)init;
-(void)showVariables;

@end
```

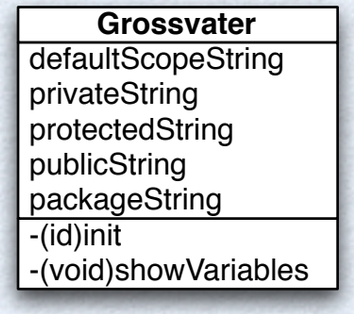
Grossvater
defaultScopeString privateString protectedString publicString packageString
-(id)init -(void)showVariables

IHM - Objective-C

El mecanismo de herencia

Archivo: Grossvater.m

```
#import "Grossvater.h"

@implementation Grossvater

-(id)init
{
    [super init];
    publicString = @"Cadena de ámbito público";
    privateString = @"Cadena de ámbito privado";
    protectedString = @"Cadena de ámbito protegido";
    packageString = @"Cadena de ámbito de paquete";
    return self;
}

-(void)showVariables
{
    NSLog(@"Grossvater -> %@", publicString);
    NSLog(@"Grossvater -> %@", privateString);
    NSLog(@"Grossvater -> %@", protectedString);
    NSLog(@"Grossvater -> %@", packageString);
}
@end
```

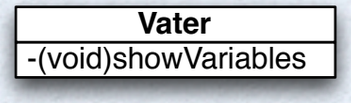


IHM - Objective-C

El mecanismo de herencia
========================

Archivo: Vater.h

```
//
//  Vater.h
//  Ambitos
//

#import "Grossvater.h"

@interface Vater : Grossvater {

}

@end
```

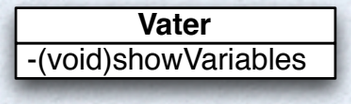


IHM - Objective-C

El mecanismo de herencia
========================

Archivo: Vater.m

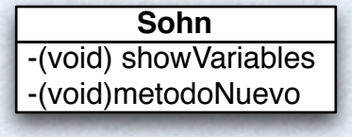
```
//
//  Vater.m
//  Ambitos
//
#import "Vater.h"

@implementation Vater

-(void)showVariables
{
    NSLog(@"Vater -> %@", publicString);
    //NSLog(@"%@", privateString);
    NSLog(@"Vater -> %@", protectedString);
    NSLog(@"Vater -> %@", packageString);
}

@end
```



IHM - Objective-C

El mecanismo de herencia
==========

Archivo: Sohn.h



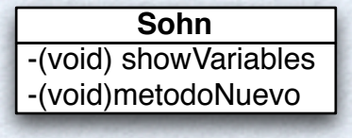

```
//
//  Sohn.h
//  Ambitos
//

#import "Vater.h"

@interface Sohn : Vater {

}

-(void)metodoNuevo;

@end
```

IHM - Objective-C

El mecanismo de herencia
==========

Archivo: Sohn.m

```
//
//  Sohn.m
//  Ambitos
//
#import "Sohn.h"
@implementation Sohn
-(void)showVariables
{
    NSLog(@"Sohn -> %@", publicString);
    //NSLog(@"%@", privateString);
    NSLog(@"Sohn -> %@", protectedString);
    NSLog(@"Sohn -> %@", packageString);
}
-(void)metodoNuevo {
    NSLog(@"Este método está definido en Sohn.");
}
@end
```



IHM - Objective-C

El mecanismo de herencia

Archivo: Ambitos.m

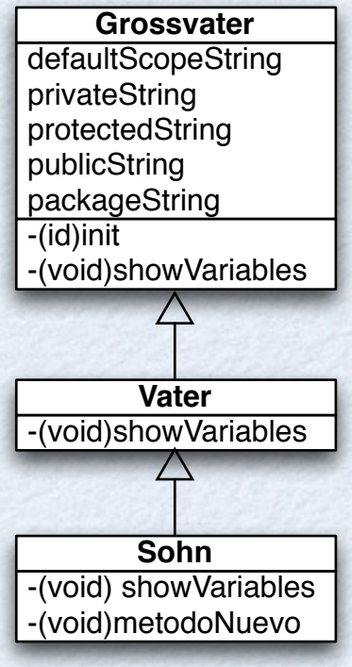
```
#import <Foundation/Foundation.h>
#import "Grossvater.h"
#import "Vater.h"
#import "Sohn.h"
#import "Schwager.h"
int main (int argc, const char * argv[]) {
    Grossvater * gv  = [[Grossvater alloc] init];
    Vater * va       = [[Vater alloc] init];
    Sohn * so        = [[Sohn alloc] init];
    Schwager * sw    = [[Schwager alloc] initWithSohn:so];
    Grossvater * test = [[Sohn alloc] init];
    Sohn * otroTest = [[Grossvater alloc] init];
    NSLog(@"Großvater:");     [gv showVariables];
    NSLog(@"Vater:");         [va showVariables];
    NSLog(@"Sohn:");          [so showVariables];
    NSLog(@"Schwager:");      [sw showVariables];
    NSLog(@"Test (Grossvater * := Sohn *)");       [test showVariables];
    NSLog(@"Otro test (Sohn * := Grosvater *)");   [otroTest showVariables];
    @try {[otroTest metodoNuevo];}
    @catch (NSException * e) {NSLog(@"El mensaje metodoNuevo no está definido en Grossvater");}
    return 0;
}
```

IHM - Objective-C

El mecanismo de herencia

Archivo: Ambitos.m

```
#import <Foundation/Foundation.h>
#import "Grossvater.h"
#import "Vater.h"
#import "Sohn.h"
#import "Schwager.h"
int main (int argc, const char * argv[]) {
    Grossvater * gv  = [[Grossvater alloc] init];
    Vater * va       = [[Vater alloc] init];
    Sohn * so        = [[Sohn alloc] init];
    Schwager * sw    = [[Schwager alloc] initWithSohn:so];
    Grossvater * test = [[Sohn alloc] init];
    Sohn * otroTest = [[Grossvater alloc] init];
    NSLog(@"Großvater:");     [gv showVariables];
    NSLog(@"Vater:");         [va showVariables];
    NSLog(@"Sohn:");          [so showVariables];
    NSLog(@"Schwager:");      [sw showVariables];
    NSLog(@"Test (Grossvater * := Sohn *)");       [test showVariables];
    NSLog(@"Otro test (Sohn * := Grosvater *)");   [otroTest showVariables];
    @try {[otroTest metodoNuevo];}
    @catch (NSException * e) {NSLog(@"El mensaje metodoNuevo no está definido en Grossvater");}
    return 0;
}
```



IHM - Objective-C

El mecanismo de herencia

Resultados de la ejecución (Está activada la recogida de basura)

```
2008-06-08 09:13:07.726 Ambitos[10812:10b] Großvater:
2008-06-08 09:13:07.728 Ambitos[10812:10b] Grossvater -> Cadena de ámbito público
2008-06-08 09:13:07.728 Ambitos[10812:10b] Grossvater -> Cadena de ámbito privado
2008-06-08 09:13:07.728 Ambitos[10812:10b] Grossvater -> Cadena de ámbito protegido
2008-06-08 09:13:07.729 Ambitos[10812:10b] Grossvater -> Cadena de ámbito de paquete
2008-06-08 09:13:07.729 Ambitos[10812:10b] Vater:
2008-06-08 09:13:07.729 Ambitos[10812:10b] Vater -> Cadena de ámbito público
2008-06-08 09:13:07.729 Ambitos[10812:10b] Vater -> Cadena de ámbito protegido
2008-06-08 09:13:07.730 Ambitos[10812:10b] Vater -> Cadena de ámbito de paquete
2008-06-08 09:13:07.730 Ambitos[10812:10b] Sohn:
2008-06-08 09:13:07.730 Ambitos[10812:10b] Sohn -> Cadena de ámbito público
2008-06-08 09:13:07.731 Ambitos[10812:10b] Sohn -> Cadena de ámbito protegido
2008-06-08 09:13:07.731 Ambitos[10812:10b] Sohn -> Cadena de ámbito de paquete
2008-06-08 09:13:07.731 Ambitos[10812:10b] Schwager:
2008-06-08 09:13:07.732 Ambitos[10812:10b] Cadena de ámbito público
2008-06-08 09:13:07.732 Ambitos[10812:10b] Cadena de ámbito de paquete
2008-06-08 09:13:07.732 Ambitos[10812:10b] Test (Grossvater * := Sohn *)
2008-06-08 09:13:07.732 Ambitos[10812:10b] Sohn -> Cadena de ámbito público
2008-06-08 09:13:07.733 Ambitos[10812:10b] Sohn -> Cadena de ámbito protegido
2008-06-08 09:13:07.733 Ambitos[10812:10b] Sohn -> Cadena de ámbito de paquete
2008-06-08 09:13:07.733 Ambitos[10812:10b] Otro test (Sohn * := Grosvater *)
2008-06-08 09:13:07.733 Ambitos[10812:10b] Grossvater -> Cadena de ámbito público
2008-06-08 09:13:07.734 Ambitos[10812:10b] Grossvater -> Cadena de ámbito privado
2008-06-08 09:13:07.734 Ambitos[10812:10b] Grossvater -> Cadena de ámbito protegido
2008-06-08 09:13:07.734 Ambitos[10812:10b] Grossvater -> Cadena de ámbito de paquete
2008-06-08 09:13:07.735 Ambitos[10812:10b] *** -[Grossvater metodoNuevo]: unrecognized selector sent to instance 0x1011950
2008-06-08 09:13:07.735 Ambitos[10812:10b] El mensaje metodoNuevo no está definido en Grossvater
```

IHM - Objective-C

El mecanismo de herencia - Conclusiones

La clase NSObject es la cabeza de jerarquía de las clases de Objective-C. No sería sencillo crear otra cabeza de jerarquía, o modificar esta.

Objective-C admite herencia simple, no herencia múltiple.

Al enviar un mensaje, se recorre la jerarquía de herencia de forma ascendente hasta hallar el método o llegar a NSObject.

A diferencia de Java, las variables protegidas son visibles en toda la cadena de herencia, y no solamente en descendientes directos.

Hay compatibilidad entre punteros de la clase base y punteros de las clases heredadas.

Cuando se invoca un método, se consulta la tabla correspondiente al puntero utilizado, independientemente del tipo de puntero en que esté almacenada esa dirección.

Se pueden redefinir métodos por herencia.

Se pueden ejecutar métodos redefinidos por herencia, empleando la expresión [super método],que da lugar a la ejecución del método perteneciente a la clase predecesora.

IHM - Objective-C

El mecanismo de creación de categorías

En ciertas ocasiones, resulta cómodo añadir métodos a una cierta clase de la que no se dispone del código fuente. O aunque se disponga del código fuente, es conveniente añadir ciertos mecanismos a la clase. Por ejemplo se podría pensar en añadir a una clase los métodos necesarios para que su estado se tome de disco o se almacene en disco de manera cómoda.

Objective-C ofrece, en este sentido, el mecanismo de creación de categorías. La creación de categorías esa análoga a la creación de clases; esto es, las categorías constan de un archivo de interfaz y de otro de implementación.

El archivo de interfaz tiene como nombre el nombre de la clase, el signo “+” y el nombre de la categoría (acabado en .h, desde luego).

Archivo: NombreDeLaClase+NombreDeLaCategoria.h”

```
#import “NombreDeLaClase.h”

@interface NombreDeLaClase(NombreDeLaCategoria)

-(tipo)metodo:(tipo)argumento...
...
-(tipo)otroMetodo:(tipo)argumento...

@end
```

IHM - Objective-C

El mecanismo de creación de categorías

El archivo de implementación tiene como nombre el nombre de la clase, el signo “+” y el nombre de la categoría (acabado en .m).

Archivo: NombreDeLaClase+NombreDeLaCategoria.m”

```
#import “NombreDeLaClase+NombreDeLaCategoria.h”

@implementation NombreDeLaClase(NombreDeLaCategoria)

-(tipo)metodo:(tipo)argumento {
...
}

-(tipo)otroMetodo:(tipo)argumento {
...
}

@end
```

No se admite la adición de variables. No se admiten múltiples métodos de igual nombre.

IHM - Objective-C

El mecanismo de creación de categorías

Como ejemplo, vamos a añadir métodos de E/S a la clase ListaDeAlimentos, con la intención de permitir que el programa cargue información de disco en el arranque, y muestre después su contenido antes de volver a almacenar la información.

La interfaz de esta categoría es como sigue:

```
//
//  ListaDeAlimentos+ES.h
//  Calorimetro
//
//  Created by coti on 08/06/08.
//  Copyright 2008 Tests by Coti. All rights reserved.
//

#import <Cocoa/Cocoa.h>
#import "ListaDeAlimentos.h"

@interface ListaDeAlimentos(ES)

    -(BOOL)readListFrom:(NSString*)theDataFile;
    -(BOOL)writeListTo:(NSString*)theDataFile;

@end
```

IHM - Objective-C

El mecanismo de creación de categorías

La implementación de esta categoría es como sigue:

```
//
//  ListaDeAlimentos+ES.m
//
#import "ListaDeAlimentos+ES.h"
@implementation ListaDeAlimentos(ES)

-(BOOL)readListFrom:(NSString*)theDataFile
{
     NSError * error = [[NSError alloc] init];
     NSString * data = [NSString stringWithContentsOfFile:theDataFile encoding:NSMacOSRomanStringEncoding error:&error];
     if (nil == data)
     {
          NSLog(@"%@", [error localizedDescription]);
          NSBeep();
          return NO;
     }
     else
     {
        ...
```

IHM - Objective-C

El mecanismo de creación de categorías
==========

La implementación de esta categoría es como sigue:

```
    ...
    }
    else
    {
        NSArray * lines = [data componentsSeparatedByString:@"\n"];
        NSArray * fields;
        const char * temp;
        for(id s in lines)
        {
            temp = [s cStringUsingEncoding:NSMacOSRomanStringEncoding];
            if (strlen(temp))
            {
                fields = [s componentsSeparatedByString:@"*"];
                if ([fields count] == 2)
                    [self registrar:[fields objectAtIndex:0] caloriasPorCienGramos:[[fields objectAtIndex:1] floatValue]];
            }
            }
            return YES;
        }
}
```

IHM - Objective-C

El mecanismo de creación de categorías

El programa principal pasa a tener este aspecto:

```
#import <Foundation/Foundation.h>
#import <AppKit/NSGraphics.h>
#import "ListaDeAlimentos.h"
#import "ListaDeAlimentos+ES.h"

int main (int argc, const char * argv[]) {
    ListaDeAlimentos * p = [[ListaDeAlimentos alloc] init];
    if (NO == [p readListFrom:@"data.txt"])
        printf("No fue posible leer el archivo data.txt");
    else
        printf("El archivo data.txt contiene %d referencias.\n", [[p lista] count]);
    NSMutableDictionary * listaAlmacenada = [p lista];

    [p registrar:@"Fresas" caloriasPorCienGramos:35.0];
    [p registrar:@"Moras" caloriasPorCienGramos:59.0];
    ...
```

IHM - Objective-C

El mecanismo de creación de categorías

El programa principal pasa a tener este aspecto:

```
    ...
    for(id alimento in listaAlmacenada)
    {
        printf("Alimento: %-25s ---> Calorías: %5.1f\n",
                        [alimento cStringUsingEncoding:NSMacOSRomanStringEncoding],
                        [[listaAlmacenada objectForKey:alimento] floatValue]);
    }

    [p writeListTo:@"data.txt"];

    return 0;
}
```

IHM - Objective-C

```
El mecanismo de creación de categorías
======================================

El resultado de la ejecución es como sigue:
 El archivo data.txt contiene 11 referencias.
 Alimento: Garbanzos                 ---> Calorías: 329.0
 Alimento: Galletas tipo María       ---> Calorías: 436.0
 Alimento: Hígado pollo, plancha     ---> Calorías: 161.0
 Alimento: Guisantes                 ---> Calorías:  67.0
 Alimento: Gambas                    ---> Calorías:  91.0
 Alimento: Habas secas               ---> Calorías: 343.0
 Alimento: Harina de trigo           ---> Calorías: 350.0
 Alimento: Berzas                    ---> Calorías:  34.5
 Alimento: Orejones                  ---> Calorías: 125.4
 Alimento: Coliflores                ---> Calorías:  64.6
 Alimento: Gallo                     ---> Calorías:  73.0
 Alimento: Moras                     ---> Calorías:  59.0
 Alimento: Fresas                    ---> Calorías:  35.0

Nota sobre la codificación y el resultado en pantalla
=====================================================

Para que %s calcule correctamente la longitud de una cadena, es necesario que esa cadena esté codificada en
MacOSRomanStringEncoding. Por tanto, he codificado todos los archivos de código fuente en MacOSRoman, y he
programado la terminal en Mac OS Roman.

El especificador de formato %25s no detecta correctamente la longitud de cadenas codificadas en UTF8.
```

IHM - Objective-C

El mecanismo de creación de categorías - Conclusiones

Objective-C permite añadir métodos a clases ya existentes, aun cuando el código fuente de esas clases no esté disponible.

Las categorías no permiten añadir variables a una clase.

Su uso permite compartimentar el desarrollo, encargando la implementación de distintas categorías a distintos miembros de un equipo. Además, las categorías facilitan la organización del código.

Las categorías están especialmente indicadas para añadir un mecanismo de E/S a estructuras de datos que inicialmente no disponen de él. Por ejemplo, es posible añadir un mecanismo de E/S en XML, o suministrar métodos que ofrezcan una codificación en UTF8 o adecuada para plataformas concretas.

Téngase en cuenta que el mecanismo de categorías no es un mecanismo de herencia. Su misión es complementar clases con métodos, no crear nuevos tipos de datos (no permite añadir atributos).

IHM - Objective-C

El mecanismo de propiedades

Los atributos de una clase requieren un cierto cuidado a la hora de definir métodos de acceso. Esta tarea no es demasiado compleja, pero si se vuelve tediosa en clases dotadas de múltiples atributos.

Las directrices de compilación @property, @synthesize y @dynamic permiten gobernar la construcción de métodos de acceso adaptados a las necesidades de ámbito y concurrencia de cada variable.

Esta sintaxis puede utilizarse en clases, categorías y protocolos.

Las propiedades ofrecen una sintaxis basada en el punto como separador; es estrictamente equivalente a la sintaxis habitual de llamada a mensajes.

IHM - Objective-C

El mecanismo de propiedades

El uso de propiedades tiene dos partes: declaración e implementación.

La parte de declaración tiene la sintaxis siguiente:

```
@property(atributos) tipo vble;
```

donde tipo denota el tipo de la propiedad, vble denota su nombre y atributos puede ser:

- readonly
- readwrite (Por defecto)
- assign (Por defecto)
- retain
- copy
- nonatomic (Por defecto los métodos de acceso son atómicos)

Obsérvese que el tipo y nombre de la variable deben coincidir con los expresados anteriormente en la zona de declaración de variables de ejemplar.

IHM - Objective-C

El mecanismo de propiedades

El uso de propiedades tiene dos partes: declaración e implementación.

La parte de implementación tiene la sintaxis siguiente:

```
@synthesize vble [,...];
```

o bien

```
@dynamic vble [,...];
```

La directriz @synthesize denota que el compilador debe crear los métodos de acceso indicados por los atributos de @property.

La directriz @dynamic indica que es responsabilidad del programador crear los métodos de acceso indicados por los atributos de @property.

IHM - Objective-C

El mecanismo de propiedades - Ejemplo

Archivo: ListaDeAlimentos.h

```
#import <Cocoa/Cocoa.h>
#import <AppKit/NSGraphics.h>

@interface ListaDeAlimentos : NSObject {
    @private
    NSMutableDictionary * lista;
    NSString * nombreDelArchivo;
}

@property(readonly) NSMutableDictionary * lista;
@property(readwrite, copy) NSString * nombreDelArchivo;

-(id)init;
-(float)caloriasPorCienGramosDe:(NSString*)alimento;
-(void)registrar:(NSString*)alimento caloriasPorCienGramos:(float)valor;
-(void)eliminar:(NSString*)alimento;

@end
```

Obsérvese que lista no se puede modificar, pero nombreDelArchivo tiene los dos métodos de acceso.

IHM - Objective-C

El mecanismo de propiedades - Ejemplo

Archivo: ListaDeAlimentos.m

```
#import "ListaDeAlimentos.h"

@implementation ListaDeAlimentos

@synthesize lista;
@synthesize nombreDelArchivo;

-(id)init {
    [super init];
    lista = [[NSMutableDictionary alloc] init];
    self.nombreDelArchivo = @"";
    return self;
}
...
```

En la parte de implementación de propiedades es imprescindible indicar que se desea que el compilador sintetice los métodos de acceso de lista (solo para leer) y nombreDelArchivo (para leer y escribir). El resto de la implementación no sufre cambios.

IHM - Objective-C

El mecanismo de propiedades - Ejemplo

Archivo: Calorimetro.m

```
int main (int argc, const char * argv[]) {
    ListaDeAlimentos * p = [[ListaDeAlimentos alloc] init];

    p.nombreDelArchivo = @"data.txt";

    if (NO == [p readListFrom:p.nombreDelArchivo])
        printf("No fue posible leer el archivo %s.\n", p.nombreDelArchivo);
    else
        printf("El archivo %s contiene %d referencias.\n",
            [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding],
            [p.lista count]);

    NSMutableDictionary * listaAlmacenada = p.lista;

    [p registrar:@"Fresas" caloriasPorCienGramos:35.0];
    [p registrar:@"Moras" caloriasPorCienGramos:59.0];

    for(id alimento in listaAlmacenada)
    {
        printf("Alimento: %-25s ---> Calorías: %5.1f\n",
            [alimento cStringUsingEncoding:NSMacOSRomanStringEncoding],
            [[listaAlmacenada objectForKey:alimento] floatValue]);
    }

...
```

IHM - Objective-C

El mecanismo de propiedades - Ejemplo
==========

Archivo: Calorimetro.m

```
    ...
    p.nombreDelArchivo = @"datosmodificados.txt";

    if (NO == [p writeListTo:p.nombreDelArchivo])
        printf("No fue posible escribir el archivo %s.\n", p.nombreDelArchivo);
    else
        printf("El archivo %s contiene ahora %d referencias.\n",
                [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding],
                [p.lista count]);

    return 0;
}
```

En la parte de implementación de propiedades es imprescindible indicar que se desea que el compilador sintetice los métodos de acceso de lista (solo para leer) y nombreDelArchivo (para leer y escribir). El resto de la implementación no sufre cambios.

IHM - Objective-C

El mecanismo de propiedades - Conclusiones

El uso de las directrices @property/@synthetize facilita la creación automática de métodos de acceso.

Es posible crear manualmente métodos de acceso empleando la directriz @dynamic.

La sintaxis de punto (destinatario.nombreDeMetodoDeAcceso, posiblemente con asignación) se asemeja más a la de otros lenguajes de programación.

Los atributos de descripción de propiedades permiten efectuar accesos no atómicos; esto tiene sentido cuando no se tiene intención de modificar el contenido de una estructura de datos desde varios hilos simultáneamente, puesto que se acelera el proceso porque no es preciso crear ni consultar semáforos.

IHM - Objective-C

El mecanismo de protocolos

Un protocolo es una colección de métodos abstractos agrupados bajo un cierto nombre. Los protocolos se corresponden directamente con las interfaces de Java.

La sintaxis de creación de un protocolo es como sigue:

```
@protocol NombreDelProtocolo <OtroProtocolo,...>

declaración de métodos abstractos

@end
```

Los protocolos no son clases y no se derivan de NSObject. Se escriben en archivos .h.

Los protocolos pueden incorporar otros protocolos, cuyos nombres se ponen a la derecha del nombre del protocolo entre corchetes angulares, separados mediante comas. Cuando una clase adopta un protocolo (lo implementa, en la terminología de Java), se indica en la forma siguiente:

```
@interface NombreDeClase : NombreDeSuperclase <NombreDeProtocolo,...>
...
@end
```

y los métodos declarados en NombreDeProtocolo deben implementarse en la parte de implementación de la clase.

IHM - Objective-C

El mecanismo de protocolos

Un cierto protocolo puede ser adoptado por tantas clases como se desee. Su misión es garantizar que las clases que lo adoptan responderán correctamente a los mensajes que define.

Las categorías también pueden adoptar protocolos, empleando una sintaxis como la siguiente:

```
@interface NombreDeClase(NombreDeCategoria) <NombreDeProtocolo,...>
// Declaración de métodos de la interfaz
@end
```

Evidentemente, la implementación de la categoría debe contener los métodos especificados en todos los protocolos indicados en la lista.

Se puede averiguar si un objeto adopta o no un cierto protocolo, empleando el método conformsToProtocol en la forma siguiente:

```
if (![objeto conformsToProtocol:@protocol(NombreDeProtocolo)]) {
    // Notificar al programador la existencia de un error
}
```

IHM - Objective-C

El mecanismo de protocolos

En principio, todos los métodos del protocolo deben ser implementados por las clases que lo adopten. Se dispone de dos directrices, @required y @optional, que permiten crear protocolos en los cuales algunos métodos sean de implementación obligatoria (@required) y otros de implementación opcional (@optional). Si no se indica nada, todos los métodos de un protocolo son obligatorios.

```
@protocol NombreDelProtocolo
    @required
    // Métodos de implementación obligatoria
    ...
    @optional
    // Métodos de implementación opcional
    ...
@end
```

IHM - Objective-C

El mecanismo de protocolos

Se pueden definir tipos y métodos que adoptan un determinado protocolo. El compilador verificará el uso correcto de esos tipos y métodos. La sintaxis es como sigue:

id <NombreDeProtocolo> nombreDeVariable;

Esta variable es el descriptor de una clase que adopta el protocolo indicado.

-(id <NombreDeProtocolo>) nombreDeMetodo;

Este método proporciona como resultado una clase que adopta el protocolo indicado.

Finalmente, se puede combinar la adopción de protocolos con el mecanismo de herencia:

NombreDeClase <NombreDeProtocolo> * descriptor;

Así se define el descriptor de un objeto que hereda de NombreDeClase y adopta el protocolo NombreDeProtocolo, sea cual fuere su implementación. Esto también se adapta al método de de funcionamiento de interfaces en Java. El compilador puede verificar que los objetos cuyo puntero se asigne a descriptor sean realmente derivados de esa clase, y que verdaderamente implementan todo los métodos definidos por el protocolo.

IHM - Objective-C

El mecanismo de protocolos

Un posible protocolo que deben implementar aquellas clases cuyo estado se pueda almacenar con formato XML podría ser el siguiente:

Archivo: BasicXML.h

```
#import <Cocoa/Cocoa.h>

@protocol BasicXML
    @required
        -(BOOL)readFromXML:(NSString *)theXMLfile;
        -(BOOL)writeToXML:(NSString *)theXMLfile;
    @optional
        -(void)allKeysUppercase:(BOOL)yesorno;
@end
```

IHM - Objective-C

El mecanismo de protocolos

Hay que indicar que ListaDeAlimentos implementa este protocolo.

Archivo: ListaDeAlimentos.h

```
#import <Cocoa/Cocoa.h>
#import <AppKit/NSGraphics.h>
#import "BasicXML.h"
/*
 Esta clase implementa el protocolo BasicXML
*/
@interface ListaDeAlimentos : NSObject <BasicXML> {
    @private
    NSMutableDictionary * lista;
    NSString * nombreDelArchivo;
}

    @property(readonly) NSMutableDictionary * lista;
    @property(readwrite, copy) NSString * nombreDelArchivo;

    -(id)init;
    -(float)caloriasPorCienGramosDe:(NSString*)alimento;
    -(void)registrar:(NSString*)alimento caloriasPorCienGramos:(float)valor;
    -(void)eliminar:(NSString*)alimento;

@end
```

IHM - Objective-C

El mecanismo de protocolos

Hay que implementar los métodos del protocolo en ListaDeAlimentos.m

Archivo: ListaDeAlimentos.m | Método: -(BOOL)readFromXML:(NSString*)theXMLfile

```
-(BOOL)readFromXML:(NSString *)theXMLfile{
    BOOL ok = YES;                  NSString *path = theXMLfile;    NSData *plistData;
    NSString *error;                NSPropertyListFormat format;    id plist;
    float tempFloat;
    plistData = [NSData dataWithContentsOfFile:path];

    plist = [NSPropertyListSerialization propertyListFromData:plistData mutabilityOption:NSPropertyListImmutable
                                                format:&format errorDescription:&error];
    if(!plist)   { NSLog(error); ok = NO;  }
    else{
        ok = YES;
        for(id clave in plist)
        {
            tempFloat = [[plist objectForKey:clave] floatValue];
            [self registrar:clave caloriasPorCienGramos:tempFloat];
        }
    }
    return ok;
}
```

IHM - Objective-C

El mecanismo de protocolos

Hay que implementar los métodos del protocolo en ListaDeAlimentos.m

Archivo: ListaDeAlimentos.m | Método: -(BOOL)writeToXML:(NSString*)theXMLfile

```
-(BOOL)writeToXML:(NSString *)theXMLfile{
    BOOL ok;
    id plist = self.lista; NSString *path = theXMLfile; NSData *xmlData; NSString *error;

    xmlData = [NSPropertyListSerialization dataFromPropertyList:plist
                                                         format:NSPropertyListXMLFormat_v1_0
                                               errorDescription:&error];
    if(xmlData)
    {
        [xmlData writeToFile:path atomically:YES];
        NSLog(@"Se han escrito los datos con formato XML en el archivo %@", theXMLfile);
        ok = YES;
    }
    else
    {
        NSLog(error);
        ok = NO;
    }
    return ok;
}
```

IHM - Objective-C

El mecanismo de protocolos
==========

Se configura el programa principal en modo de lectura o escritura

Archivo: ListaDeAlimentos.m | Modo: Lectura

```
p.nombreDelArchivo = @"datossalida.xml";

if (NO == [p readFromXML:p.nombreDelArchivo])
      printf("No fue posible leer el archivo XML %s.\n", p.nombreDelArchivo);
else
      printf("El archivo %s contiene %d referencias.\n",
                  [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding], [p.lista count]);

for(id alimento in listaAlmacenada)
{
      printf("Alimento: %-25s ---> Calorías: %5.1f\n",
                  [alimento cStringUsingEncoding:NSMacOSRomanStringEncoding],
                  [[listaAlmacenada objectForKey:alimento] floatValue]);
}
```

IHM - Objective-C

El mecanismo de protocolos

Se configura el programa principal en modo de lectura o escritura

Archivo: ListaDeAlimentos.m | Modo: Escritura

```
p.nombreDelArchivo = @"data.txt";
    if (NO == [p readListFrom:p.nombreDelArchivo])
        printf("No fue posible leer el archivo %s.\n", p.nombreDelArchivo);
    else
        printf("El archivo %s contiene %d referencias.\n",
            [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding], [p.lista count]);
[p registrar:@"Fresas" caloriasPorCienGramos:35.0]; [p registrar:@"Moras" caloriasPorCienGramos:59.0];
for(id alimento in listaAlmacenada)
{
    printf("Alimento: %-25s ---> Calorías: %5.1f\n",
            [alimento cStringUsingEncoding:NSMacOSRomanStringEncoding],
            [[listaAlmacenada objectForKey:alimento] floatValue]);
}

p.nombreDelArchivo = @"datosmodificados.txt";

if (NO == [p writeListTo:p.nombreDelArchivo])
    printf("No fue posible escribir el archivo %s.\n", p.nombreDelArchivo);
else
    printf("El archivo %s contiene ahora %d referencias.\n",
            [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding], [p.lista count]);

[p writeToXML:@"datossalida.xml"];
```

IHM - Objective-C

```
El mecanismo de protocolos
==========================

Se configura el programa principal en modo de lectura o escritura

Resultados de la ejecución (lectura de XML).

El archivo datossalida.xml contiene 13 referencias.
Alimento: Garbanzos                  ---> Calorías: 329.0
Alimento: Galletas tipo María        ---> Calorías: 436.0
Alimento: Guisantes                  ---> Calorías:  67.0
Alimento: Gambas                     ---> Calorías:  91.0
Alimento: Habas secas                ---> Calorías: 343.0
Alimento: Harina de trigo            ---> Calorías: 350.0
Alimento: Berzas                     ---> Calorías:  34.5
Alimento: Orejones                   ---> Calorías: 125.4
Alimento: Coliflores                 ---> Calorías:  64.6
Alimento: Gallo                      ---> Calorías:  73.0
Alimento: Moras                      ---> Calorías:  59.0
Alimento: Hígado de pollo,plancha    ---> Calorías: 161.0
Alimento: Fresas                     ---> Calorías:  35.0
```

IHM - Objective-C

El mecanismo de protocolos - Conclusiones

Los protocolos de Objective-C se corresponden con las interfaces de Java.

Sirven para garantizar que ciertas clases van a responder correctamente a determinados mensajes.

Se admite que una clase adopte (Objective-C) o implemente (Java) más de un protocolo.

La equivalencia sintáctica entre Java y Objective-C es asi:

```
//Java

class NombreDeClase extends Superclase implements Protocolo {...

// Objective-C

@interface NombreDeClase : SuperClase <Protocolo> ...
```

IHM - Objective-C

El mecanismo de tratamiento de excepciones

Básicamente es análogo al mecanismo ofrecido por Java; tiene la sintaxis siguiente:

```
@try {
    // Código que puede lanzar una excepción

}
@catch (NSException * e) {
    // Tratamiento de la excepción

}
@finally {
    // Código que se ejecutará tanto si hubo una excepción como si no.

}
```

El funcionamiento de este mecanismo es análogo al de Java: toda excepción que se produzca en el bloque @try dara lugar a la ejecución del primer bloque @catch que tenga como variable de control una excepción compatible.

IHM - Objective-C

El mecanismo de tratamiento de excepciones

Como ejemplo, véase el programa anterior modificado para utilizar excepciones.

Archivo: Calorimetro.m | Configuración de lectura

```
    @try {
        p.nombreDelArchivo = @"datossalida.xml";
        //p.nombreDelArchivo = @"cajones.xml";
        [p readFromXML:p.nombreDelArchivo];
        printf("El archivo %s contiene %d referencias.\n",
                  [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding],
                  [p.lista count]);
    }
    @catch (NSException * e) {
        printf("No fue posible leer el archivo XML %s.\n", p.nombreDelArchivo);
        NSLog(@"%@ : %@", [e name], [e reason]);
    }
    @finally {
        for(id alimento in listaAlmacenada)
        {
            printf("Alimento: %-25s ---> Calorías: %5.1f\n",
                [alimento cStringUsingEncoding:NSMacOSRomanStringEncoding],
                [[listaAlmacenada objectForKey:alimento] floatValue]);
        }
    }
```

IHM - Objective-C

El mecanismo de tratamiento de excepciones

Como ejemplo, véase el programa anterior modificado para utilizar excepciones.

Archivo: Calorimetro.m | Configuración de escritura

```
@try{    p.nombreDelArchivo = @"data.txt"; [p readListFrom:p.nombreDelArchivo];
         printf("El archivo %s contiene %d referencias.\n",
               [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding], [p.lista count]);
}
@catch(NSException * e)    {   NSLog(@"%@ : %@", [e name], [e reason]); }
@finally {[p registrar:@"Fresas" caloriasPorCienGramos:35.0]; [p registrar:@"Moras" caloriasPorCienGramos:59.0];
      for(id alimento in listaAlmacenada)
      {   printf("Alimento: %-25s ---> Calorías: %5.1f\n",
               [alimento cStringUsingEncoding:NSMacOSRomanStringEncoding],
               [[listaAlmacenada objectForKey:alimento] floatValue]);
      }
}
@try {    p.nombreDelArchivo = @"datosmodificados.txt"; [p writeListTo:p.nombreDelArchivo];     }
@catch (NSException * e) { NSLog(@"Error: %@ : %@", [e name], [e reason]); }
@finally {  printf("El archivo %s contiene ahora %d referencias.\n",
               [p.nombreDelArchivo cStringUsingEncoding:NSMacOSRomanStringEncoding], [p.lista count]);
}
```

IHM - Objective-C

El mecanismo de tratamiento de excepciones

Procede comentar que Cocoa ofrece un mecanismo para el tratamiento de errores similar al de Java, pero basado en otra clase denominada NSError. Los métodos de los distintos frameworks de Cocoa responden con nil o NO cuando se produce algún error de ejecución, y además devuelven (por referencia) un ejemplar personalizado de NSError.

El método que recibe nil o NO al efectuar una llamada accede a ese ejemplar de NSError y actúa en consecuencia. Se recomienda aprovechar el mecanismo de NSError en frameworks ya construidos. El mecanismo de excepciones también puede emplearse, especialmente en módulos cerrados en los que pueda resultar más cómodo, quizá por tratarse de software procedente de otros entornos en que el mecanismo de excepciones está muy extendido (Java o C++, por ejemplo).

IHM - Objective-C

Comentarios finales

Objective-C es un lenguaje orientado a objetos con características similares a las de Java (del cual es predecesor).

Se dispone de un mecanismo de herencia simple (no múltiple).

Se dispone de un mecanismo de extensión mediante categorías.

Se dispone de un mecanismo de extensión mediante protocolos (interfaces).

Se dispone de un mecanismo de generación automatizada de métodos de acción para propiedades

Se dispone de un mecanismo de recolección automática de basura (de hecho hay otro manual que no se ha mencionado).

Es posible (habitual) mezclar Objective-C con C, e incluso con C++ (con ciertas precauciones).

IHM - Objective-C

El paso siguiente --->
==========

Objective-C posee una extensa colección de clases orientadas a la constructor de interfaces gráficas de usuario. Esta colección (Cocoa) se maneja desde Objective-C con la ayuda de otro programa, Interface Builder, que permite construir muy sofisticadas interfaces gráficas de usuario.

Texto

Universidad de Salamanca

IHM - OBJECTIVE-C

Depto. Informática y Automática
Máster en Sistemas Inteligentes
Dr. J.R. García-Bermejo Giner