Programación Orientada a Objetos con Cocoa

Objective C

Amilcar Meneses Viveros

Sección Computación
Departamento de Ingeniería Eléctrica
CINVESTAV-IPN México, D.F.
(2003)

Un poco de historia...

- Desarrollado por Brad Cox en 1983
- Se funda Stepstone Corp.
- 1985 - Steve Jobs deja Apple e inicia NeXT
  - Se elige Objective C como lenguaje para desarrollar aplicaciones en NeXT
- 1989 - Se libera NeXTSTEP 1.0
- 1991 - Los cambios de Objective C se hacen desde gcc
- NeXTSTEP 3.1 tiene una versión para PC
  - Versión robusta de objetos distribuidos
  - Contiene compilador y depurador de Objective C
- 1995 - NeXT adquiere todos los derechos de Objective C a Stepstone
- 1997 - Apple adquiere NeXT (Steve regresa a casa) e inicia el trabajo de Mac OS X.
- 2001 - Apple libera la versión 10.1 de OS X ...

Características de Objetive C

Basado en C A diferencia de C++, Objective C es un superconjunto de C. Agrega a la sintáxis de C, la manera de enviar mensajes en Small-Talk y de definir e implantar los objetos.

Run Time System Este sistema auxiliar le permite hacer la tipificación dinámica y el ligado dinámico.

Rápido Objective C ejecuta rápidamente el llamado de funciones (de 1.5 a 2.0 del llamado de una función), además que permite tener una tipificación estática si se requiere. Y la persistencia de objetos se maneja por número de referencia (se evita el manejo de un colector de basura que es costozo en el rendimiento de las aplicaciones).

Tipos de datos

- Utiliza los datos aceptados por C.
- Utiliza el tipo `id` como el identificador para cualquier tipo de objeto (tipificación dinámica).

```
id unObjeto, otroObjeto;
```

- Cuando se declara un objeto tiene por omisión el valor nulo `nil`.

```
id unObjeto;
unObjeto = nil;
```

- Cuando se declara un identificador como un apuntador de un objeto de alguna clase, el compilador realiza una tipificación estática.

```
NSString *cadena;
```

Clases

- Objetive C utiliza jerarquía sencilla.
- Se hace una división entre la *Interfaz* (`.h`) y la *Implantación* (`.m`).
- Interfaz (`Clase.h`)

```
#import ``SuperClase.h''
@interface Clase : SuperClase
{
   // Declaracion de variables de la instancia
}
// Declaracion de metodos (de clase y de instancia)
@end
```

- Métodos de la instancia:

```
- (tipo)metodoInstancia;
```

- Métodos de la clase:

```
+ (tipo)metodoClase;
```

Clases

- Implantación (`Clase.m`)

```
#import ``Clase.h''

@implementation Clase
\\ Implantacion de los metodos de la instancia y
\\ de la clase
@end
```

- Método de intancia

```
- (tipo)metodoInstancia
{
   tipo a;
   ...
   return a;
}
```

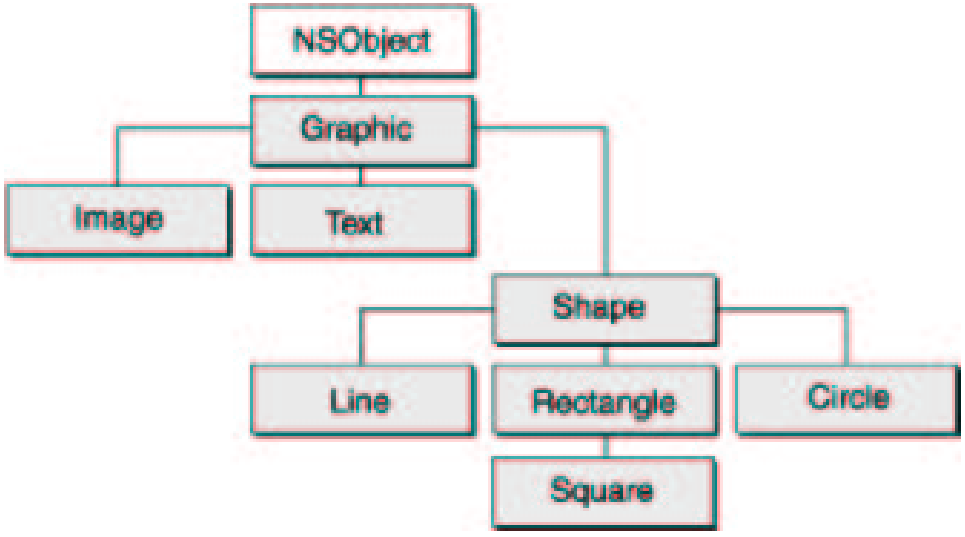
Clases

- Métodos de la clase y “¿variables de la clase?”

```
#import ``Clase.h''

tipo varClase;

@implementation

+ (tipo)metodoClase
{
  ...
  return varClase;
}

@end
```

Herencia

- Es el mecanismo para aprovechar la reutilización de código.
- Objective C maneja herencia sencilla



Herencia

```
@interface Person : NSObject
{
   NSString *nombre;
   NSDate   *fechNacimiento;
   Addres   *direccion;
   id       rfc;
   id       edoCivil;
}
- (id)initWithName: (id) nombre;
- (id)ponDireccion: (id) dir;
- (id)ponRfc: (id) nrfc;
- (id)ponEdoCivil: (id)civil;
- (int)edad;
- (id)rfc;
@end
```

```
#import ``Persona.h''

@interface Trabajador : Persona
 {
    id    puesto;
    int   clave;
    float salario;

 }
- (id) ponPuesto: (id) npuesto;
- (id) ponPuesto: (int) nclave;
- (id) ponSalario: (float)nsal;
- (id) puesto;
- (int) clave;
- (float) salario;
@end
```

Alcance de variables

- Variables declaradas en un método solo son visibles en el mismo.
- Variables declaradas como clase, son visibles solo en los métodos (de la clase e instancia) de los objetos de esa clase y de los objetos de las subclases.
- Variables de instancia pueden ser privadas, protegidas y públicas.

  `Private` Sólo los objetos de la clase donde se declara.

  `Protected` Se trabajan en los objetos de la clase y de las subclases. Por omision las variables son protegidas.

  `Public` Se pueden alcanzar desde otro objeto a través del identificador (no es recomendable, ya que viola el encapsulamento).

Alcance de variables

```
@interface Persona : NSObject
{
   NSString *nombres;
   NSDate   *fechNacimiento;
   NSString *curp;
   id       domicilio;
   id       numTel;
@private
   int      edad;
@ public
   Sexo     sexo;
@protected
}
@end

Persona *p=[[Persona alloc] init];
p->sexo=cambioSexo(); // Cuidado!! Se viola el encapsulamiento
                      // Debe ser tipo estatico
```

Mensajes

La manera de llamar mensajes es:

```
[receptor mensaje];
nombre = [administrador nombre];
edad   = [administrador edad];
[adminstrador nuevoSalario: 20000];
```

Los mensajes permiten obtener información del estado de un objeto.

Los mensajes permiten que un objeto ejecute algun acción.

Los mensajes permiten modificar el estado de un objeto.

Mensajes

En Objective C los mensajes se ligan en tiempo de ejecución. El compilador convierte la expresion:

```
[receptor mensaje];
```

a un llamado a la función `objc_msgSend()`

```
objc_msgSend(receptor, selector)
```

Los mensajes pueden tener multiples argumentos:

```
objc_msgSend(receptor, selector, arg1, arg2, ...)
```

Un selector identifica el nombre de un método.

Mensajes

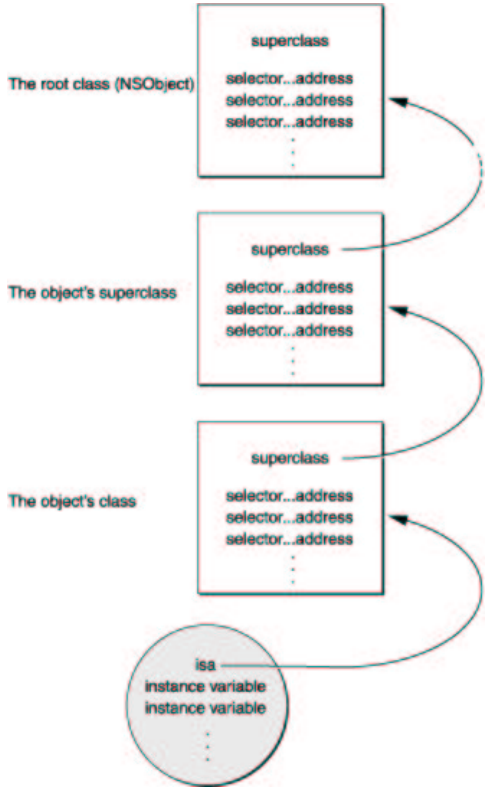


Figura tomada de la documentación de Apple.

Mensajes

Se pueden enviar mensajes a `self` y a `super`.

Los selectores pueden manejarse por el usuario

```
SEL miSelector;
miSelector = @selector(setWidth:height:)
```

El `selector` identifica el nombre de un método. Cuando se trabaja con el polimorfismo varias clases pueden tener el mismo `selector`, pero diferentes implantaciones del método.

Mensajes

Se puede cambiar el mensaje en tiempo de ejecución. Los metodos `performSelector:`, `performSelector:withObject:`, y `performSelector:withObject:withObject:` toman el identificador `SEL`como su argumento inicial.

```
[operadora perfotmSelector: @selector(atiendeLlamadaDe:)
    withObject: cliente];
```

Esto es equivalente a tener:

```
[operadora atieneLlamadaDe: cliente];
```

Estos metodos hacen posible el cambiar el mensaje en tiempo de ejecución.... ¡¡y el objeto receptor también!!...

```
id   delegado  = getReceptor();
SEL  solicitud = getSelector();
[delegado performSelector:solicitud];
```

Paradigma Sender-Target

El `Application Kit` de Cocoa explota las facilidades de modificar el receptor y el mensaje en tiempo de ejecución.

Cuando se desarrollan aplicaciones basadas en Cocoa, la construccion del la interfaz gráfica de usuario

Los objetos gráficos que se utilizan para desarrollar una GUI de una aplicación basada en Cocoa (objetos NSControl), permiten definirles el objeto al que le mandaran un mensaje y el selector. Esto se hace graficamente con el `InterfaceBuider`, pero esto puede codificarse de la siguiente manera:

```
[miBoton setAction:@selector(grafica:)];
[miBoton setTarget: graficador];
```

Categorías

- Es una alternativa a las subclases. La categoría agrega nuevos métodos a la clase.
- La categoría solo agrega métodos, no variables instancia (¡lo que se extiende es la interfaz!).
- Como el caso de la subclase, no es necesario el código de la clase que se extiende.
- Las subclases heredan los métodos agregados por categorías de la super clase.

Categorías

```
/* MiCategoria.h */

#import ``UnaClase.h''

@interface UnaClase (MiCategoria)

// Definicion de metodos

@end
```

```
/* MiCategoria.m */

#import ``MiCategoria.h''

@implementation UnaClase (MiCategoria)

// Implantacion de los metodos

@end
```

Protocolos

- Es la declaración de una interfaz para que alguien más la implante.
- Permite que diferentes familias de clases puedan responder a un mismo conjunto de métodos.
- Para declarar un protocolo

```
@protocol NombreProtocolo
// Declaracion de metodos
@end
```

- Para agregar un protocolo a una clase:

```
@interface UnaClase : LaSuperclase < protocolo1, protocolo2, ... >
```

- Para agregar un protocolo a una categoria:

```
@interface UnaClase (MiCategoria) < protocolo1, protocolo2, ...>
```

Bibliografía

1. Apple Documentation; *"The Objective-C Languaje Programing"*; Apple Inc.; Feb. 2003.
2. Brad Cox; *"Object-oriented Programming, An Evolutionary Approach"*; Addison Wesley Publishing Company, Second Edition 1986.
3. Edward Dale & Tim Watts; *"Objective C"*; `http://www.cs.rit.edu/~ats/plc-2002-2/reports/objc/intro.html` 2002.
4. *"Wikipedia"*; `http://www.wikipedia.org/` 2001.