Tecnologías para el desarrollo de aplicaciones en dispositivos móviles

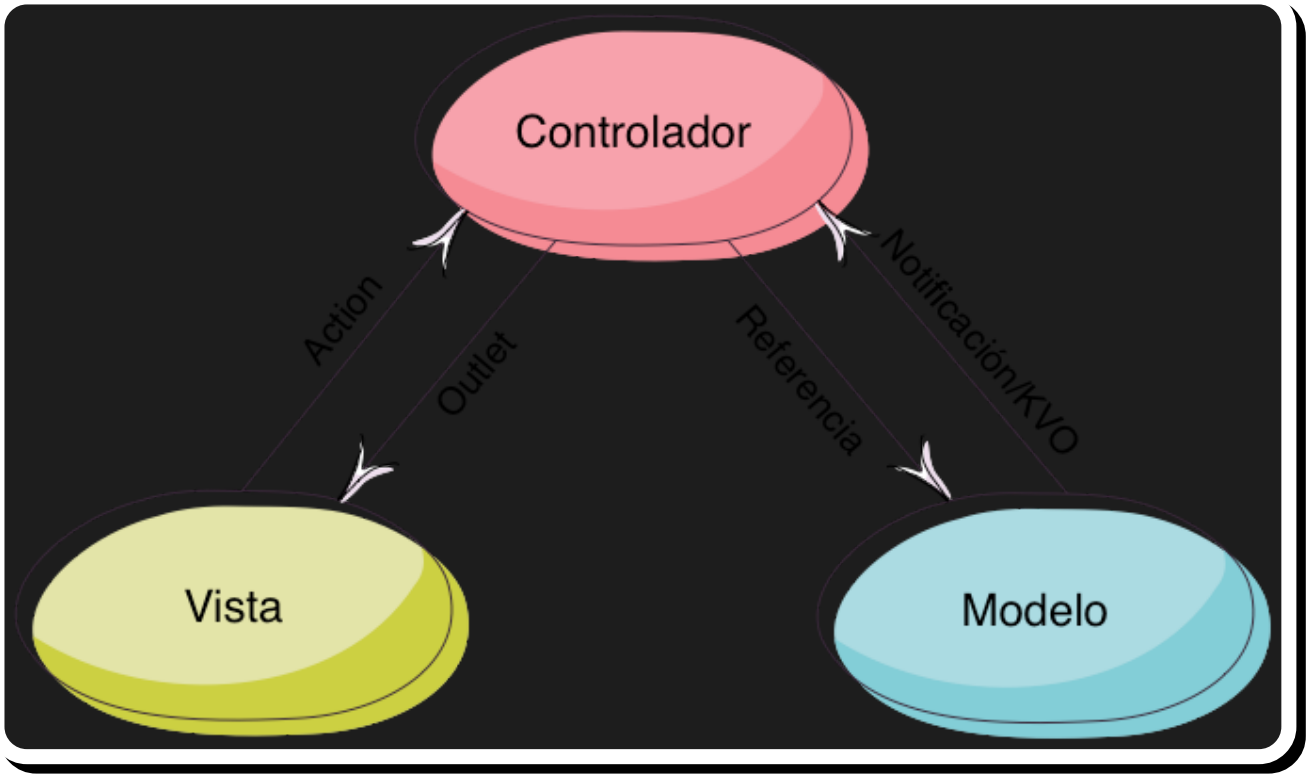
Sesiones 1 y 2: hola iOS

Parte 2: Introducción al desarrollo iOS en Objective-C

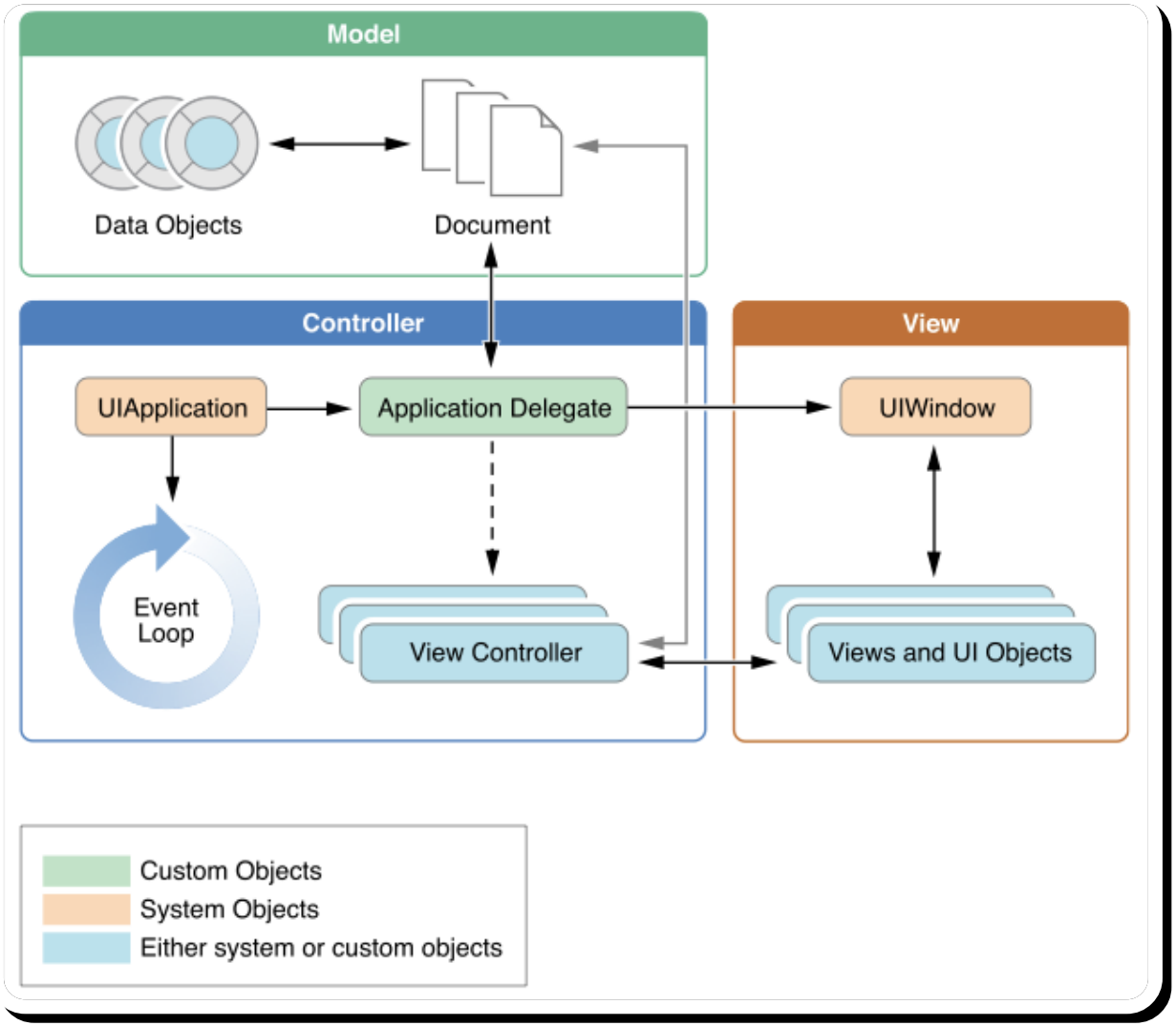
Puntos a tratar

- Estructura de las aplicaciones iOS
- Introducción a Objective-C
- El *framework* `Foundation`

Modelo/Vista/Controlador



Estructura de una aplicación iOS



Un poco de historia

- Creado en 1980 por Brad Cox y Tom Love, inspirado en Smalltalk
- Adoptado por NeXT como lenguaje de desarrollo en 1988
- En 1996 Apple compra NeXT y usa NexTSTEP como base para OS X
- En 2006 se presenta Objective-C 2.0, una modernización de la sintaxis
- En 2014 Apple presenta Swift como alternativa a Obj-C

Algunas características del lenguaje

- Es una **extensión de C** orientada a **objetos**
- Es mucho más **dinámico** que C++. Por ejemplo, se puede decidir el método a llamar en tiempo de ejecución.
- Todos los objetos son **referencias**, al estilo Java, pero debemos explicitarlo.

```
//Obj-C
NSString *cadena = @"hola";
//Esto no es legal
NSString cadena2;
//Java
String cadena = "hola";
```

Algunas características del lenguaje (2)

- No se usa la sintaxis OO clásica de **`objeto.metodo(parametro)`** sino otra mucho más atípica: **`[objeto metodo: parametro]`**

```
UIAlertView *alert = [[UIAlertView alloc]
                          initWithTitle:titulo
                          message:mensaje
                          delegate:self
                          cancelButtonTitle:@"OK"
                          otherButtonTitles: nil];
[alert show];
```

Algunas características del lenguaje (3)

- La **interfaz** (archivo `.h`) está separada de la **implementación** (`.m`)
- Se usan `#import`, muy parecidos a los `#include` de C.
- La gestión de memoria se puede **automatizar**, aunque no usa recolección de basura, sino *cuenta de referencias*

Introducción a Obj-C con una aplicación de ejemplo

Vamos a desarrollar una aplicación llamada **UADivino** que básicamente es una versión simplificada de la típica "bola 8"

Le formulamos a la app una pregunta "en voz alta" y nos responde sí/no

¿Qué es Foundation?

- Conjunto de clases básicas y utilidades que no existen directamente en Obj-C
  - Colecciones, cadenas, fechas, ...
  - Clase raíz de la jerarquía de clases: `NSObject`
  - Tipos primitivos para mejorar portabilidad

Tipos primitivos de Foundation/Cocoa

- Definidos simplemente con `typedef`, su objetivo es mejorar la portabilidad
- `NSInteger` y `NSUInteger` como sustitutos de `int` y `unsigned int`
  - Definidos como 32/64 bits según la versión de la plataforma
- `CGFloat`: sustituto para `float/double`

Clases mutables e inmutables

- Ciertas clases de Foundation/Cocoa son **inmutables**, una vez instanciado un objeto no se puede modificar. Ejemplo: **`NSString`**. Otras son **mutables**.
- Para cada clase básica (cadena, lista, diccionario, ...) generalmente tenemos versión inmutable (NSXxx...) y mutable (NSMutableXxx)

```
//NSString es inmutable
NSString *mensaje = @"Hola";
NSMutableString *mensaje2 = [[NSMutableString alloc] initWithCapacity:10];
[mensaje2 appendString:@" mundo"];
```

NSObject

- Es la raíz de la jerarquía de clases
- Todas las clases deben heredar de ella

```
@interface MiClase : NSObject

@end
```

`NSObject`: métodos para copia de objetos

```
// La cadena original es inmutable
NSString *cadena = @"Mi cadena";
NSString *copiaInmutable = [cadena copy];
NSMutableString *copiaMutable = [cadena mutableCopy];
```

- Podemos hacer que **`copy`** funcione con nuestras clases implementando algunos métodos

`NSObject`: información sobre una instancia

- **Sobreescribiremos** estos métodos en nuestras clases. Las clases de Foundation/Cocoa ya lo hacen.
- `isEqual`: comprobar igualdad entre clases. Como el `equals` de Java

```
//Esto no va a ser cierto, '==' comprueba igualdad de referencias
if (@"hola"==@"hola")
   NSLog(@"Mal");
//Esto sí es correcto
if ([@"hola" isEqual:@"hola"])
   NSLog(@"OK");
```

`NSObject`: info. sobre una instancia (2)

- `description`: debe devolver un `NSString*` con una descripción legible. Idem al `toString` de Java.

```
NSDate *fecha = [[NSDate alloc] init];
NSLog(@"%@", [fecha description]); //2014-10-11 13:52:44 +0000
```

- `hash`: dos objetos iguales deberían tener el mismo valor (un `NSUInteger`)

Colecciones

- Todas las colecciones de Foundation vienen en una versión "inmutable" y otra "mutable".
- Cuando se añade un objeto a una colección mutable lo que se está añadiendo es una referencia, no una copia

Wrappers de tipos primitivos

- Las colecciones son conjuntos de **objetos**, por lo que directamente no pueden almacenar valores **primitivos** (`int`, `float`,...) . Tenemos que "empaquetarlos" en objetos usando *wrappers*.
- `NSNumber` es un *wrapper* para datos numéricos

```
//definimos un NSNumber precediendo un literal de '@' (similar a cadenas)
NSNumber *num = @7.25;
float flo = 3.5;
//Si es una expresión necesitamos paréntesis
NSNumber *otro_num = @(flo);
```

Listas

- Versión inmutable: `NSArray` , la mutable es su subclase `NSMutableArray`
- Un `NSArray` puede contener objetos de distintas clases. Además, los `NSMutableArray` pueden cambiar no solo de **contenido** sino también de **tamaño**.

Inicialización de listas

- Lo más sencillo es con literales para las inmutables. Para las mutables necesitamos métodos

```
//Obsérvese que contiene distintas clases
NSArray *a = @[@1, @"hola"];
NSMutableArray *mut = [NSMutableArray arrayWithObjects:
                                    @1, @"Hola", nil];
```

Listas: algunos métodos útiles

```
NSMutableArray *mut = [NSMutableArray arrayWithObjects:
                                @"Alo", @"Hola", nil];
//Insertar en una posición. Los existentes se "desplazan" a la derecha
[mut insertObject:@"Hello" atIndex:0];
//cuántos elementos tiene
[mut count]
//Eliminar todas las apariciones de un objeto
[mut removeObjectIdenticalTo:@"Hola"];
//Se puede usar notación "clásica" de array
NSLog(@"%@", mut[0]);
//los predicados nos permiten expresar condiciones para filtrar datos
NSPredicate *comienza_por_h = [NSPredicate predicateWithFormat:@"SELF beginswith[c] '
//ahora contendrá solo @"Hello"
[mut filterUsingPredicate:comienza_por_h];
```

Diccionarios

- Conjunto de pares `(clave, valor)`. Las claves suelen ser `NSString`, pero se puede usar cualquier clase copiable (conforme a `NSCopying`) e implemente `isEqual` y `hash`.
- `NSDictionary` es la versión inmutable y `NSMutableDictionary` la mutable.

Diccionarios: algunos métodos útiles

```
//Inicializar inmutables con literales
NSDictionary meses = @{
    @"Enero" : @31,
    @"Febrero" : @28,
    @"Marzo" : @30
}
//Inializar mutables con método factory
//Varias posibilidades, por ejemplo a partir de arrays con valores y claves
NSArray *nombres = @[@"Enero", @"Febrero", @"Marzo"];
NSArray *dias = @[@31, @28, @30];
NSMutableDictionary *meses = [NSMutableDictionary dictionaryWithObjects:dias
                                                                forKeys:nombres];
//Para añadir podemos usar notación de "array" con la clave...
meses[@"Septiembre" ] = @30;
//... y también usar métodos
[meses setObject:@31 forKey:@"Diciembre"];
```

Recorrer colecciones

- *Fast enumeration*: es un **`for-in`** al estilo Java.

```
//Como no sabemos qué clases puede haber, usamos el tipo genérico: id
for(id obj in coleccion) {
    NSLog(@"Obtenido el objeto %@", obj);
}
//Si conocemos el tipo podemos especificarlo
for(NSString *cad in coleccion_palabras) {
    NSLog(@"Obtenida la cadena %@", cad);
}
//Aplicado a un diccionario, itera por las claves
for (id clave in diccionario) {
    NSLog(@"(%@, %@)", clave, [diccionario objectForKey: clave])
}
```

- *Enumerators*: siguen el patrón "Iterator". Código menos legible.