Desarrollo de Aplicaciones iOS

Sesión 4: Vistas

Puntos a tratar

- Patrón MVC
- Jerarquía de vistas
- Creación de vistas con Interface Builder
- Creación de vistas de forma programática
- Propiedades de las vistas
- Controles básicos

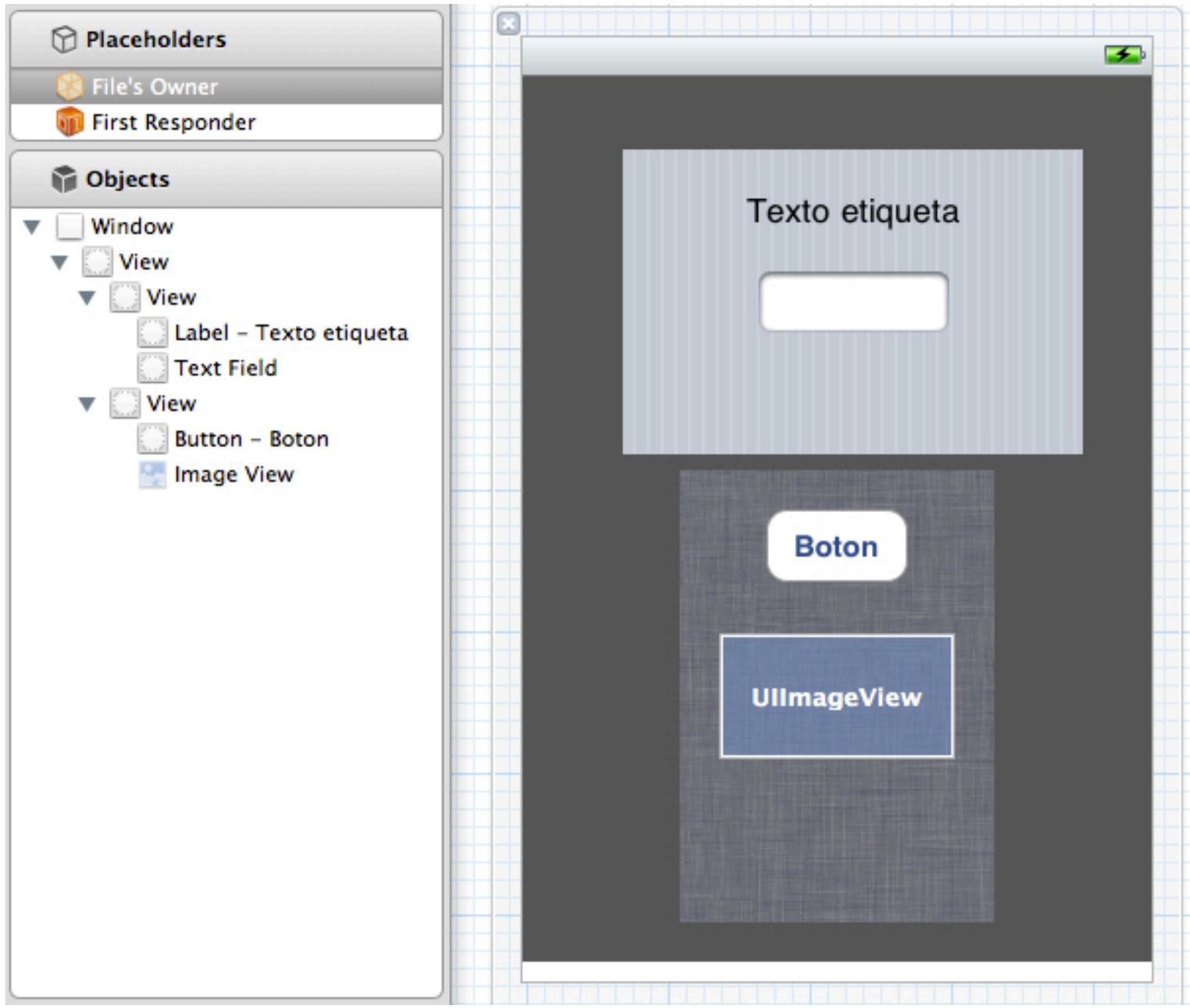
Patrón MVC

- La API de Cocoa Touch está basada en patrón MVC
  - Modelo
    - Objetos de dominio
    - Operaciones de negocio
    - Acceso a datos
  - Vista
    - Interfaz de la aplicación
    - Clases que derivan a `UIView`
    - Podemos crear vistas de forma visual con *Interface Builder*
  - Controlador
    - Coordina modelo y vista
    - Es propio de la aplicación (no reutilizable)
    - Clases que derivan de `UIViewController`

Jerarquía de vistas

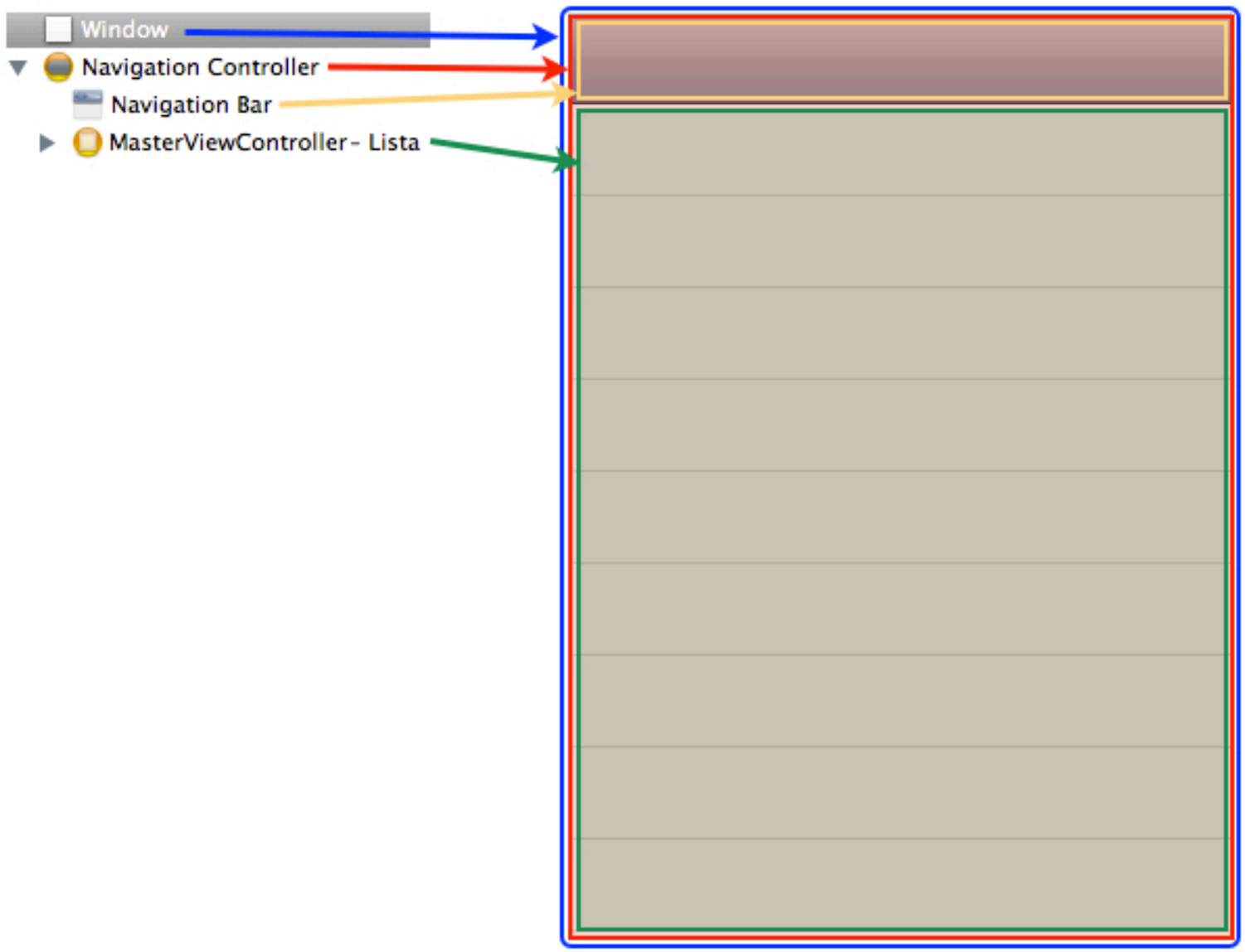
- Ventana
  - `UIWindow`
  - Una por aplicación
  - Contiene vistas
- Vistas
  - `UIView`
  - Definen un marco
  - Dibujan contenido
  - Responden ante eventos

    `UIResponder`
  - Contiene subvistas
  - Diferentes subtipos



Controladores

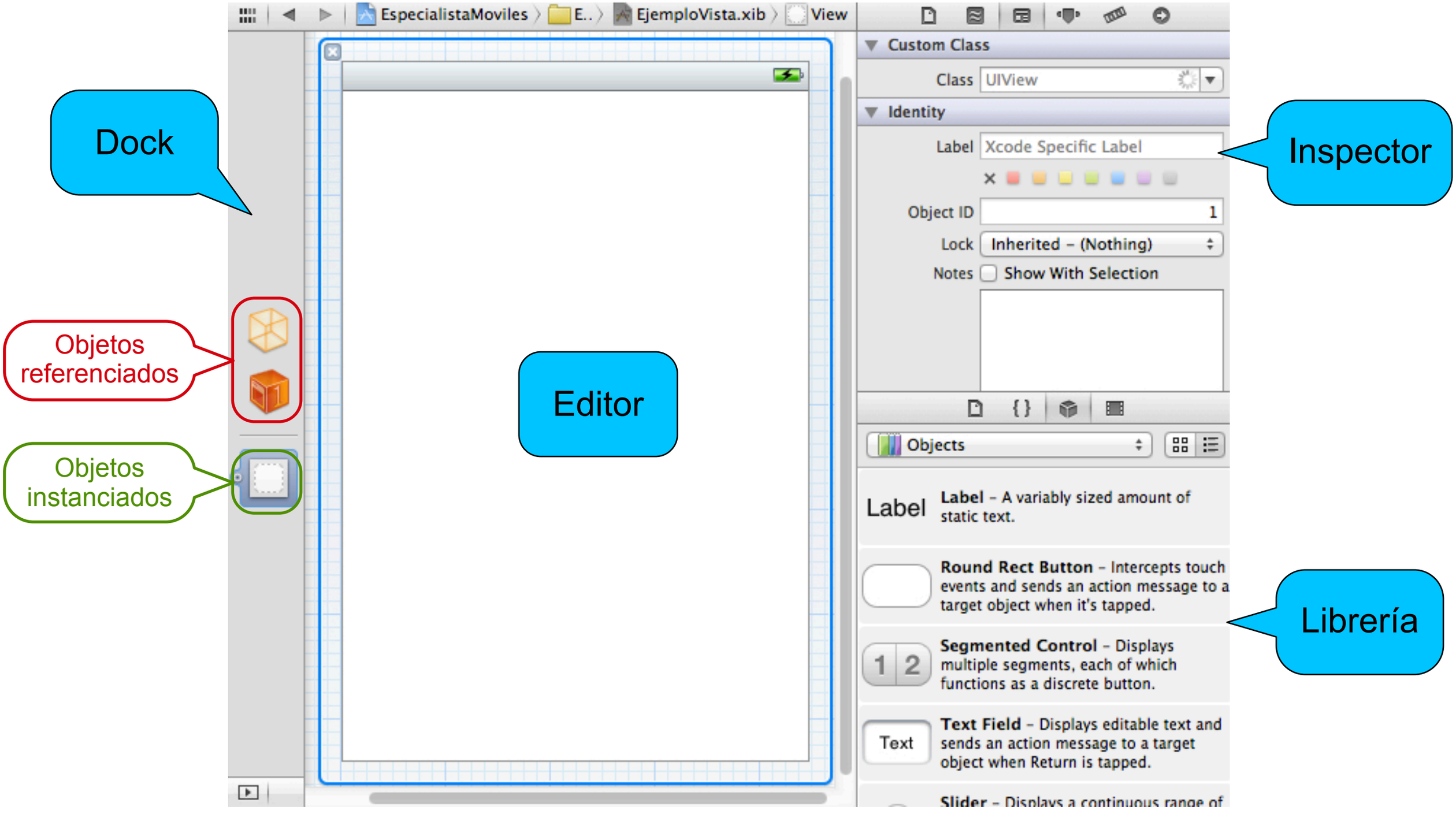
- Normalmente las vistas están gestionadas por controladores



Interface Builder

- Crea la interfaz de forma visual
- Los objetos de la interfaz se guardan en un fichero `.xib`
  - Al compilarse se convierte en un fichero `.nib`
  - Es un fichero contenedor de objetos creados de antemano
  - Puede incluir referencias a objetos externos ya existentes
- Creamos un fichero NIB con
  - *File > New > New File ... > iOS > User Interface*
  - Podemos elegir entre diferentes plantillas

Entorno de Interface Builder



Objetos referenciados

- No son instanciados por el NIB, existen de antemano
  - *File's Owner*

    Al cargar un NIB siempre hay que especificar un objeto que haga de propietario

    Puede ser cualquier tipo de objeto

```
[[NSBundle mainBundle] loadNibNamed: @"NombreNib"
                              owner: self
                            options: nil];
```

    Podremos conectar elementos instanciados por el NIB con propiedades del *File's Owner*

  - *First Responder*

    Su valor es `nil`

Outlets

- Los *outlets* conectan el código de nuestra aplicación con los objetos instanciados por el NIB
  - Se definen como propiedades de tipo `IBOutlet`

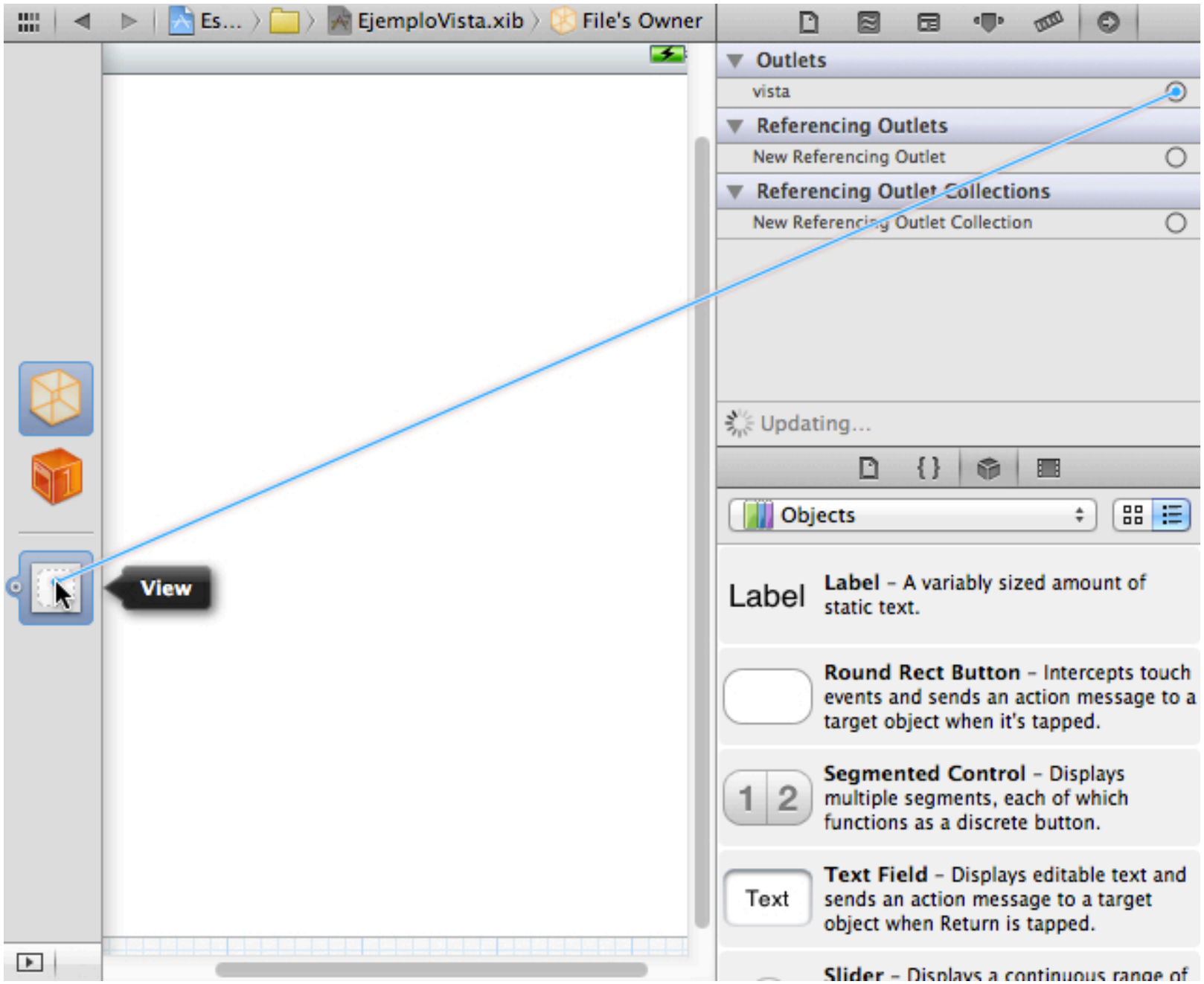
Se deben retener sólo los objetos raíz

```
@interface EjemploVista : NSObject

@property(nonatomic,strong) IBOutlet UIView *vista;

@end
```

- El tipo `IBOutlet` se elimina en preprocesamiento
  - Sólo se utiliza para que el entorno los reconozca como tales

- Estableceremos la conexión desde el inspector
  - En el inspector de identidades debemos especificar el tipo correcto para el *File's Owner*

Conexión con outlets



Acciones

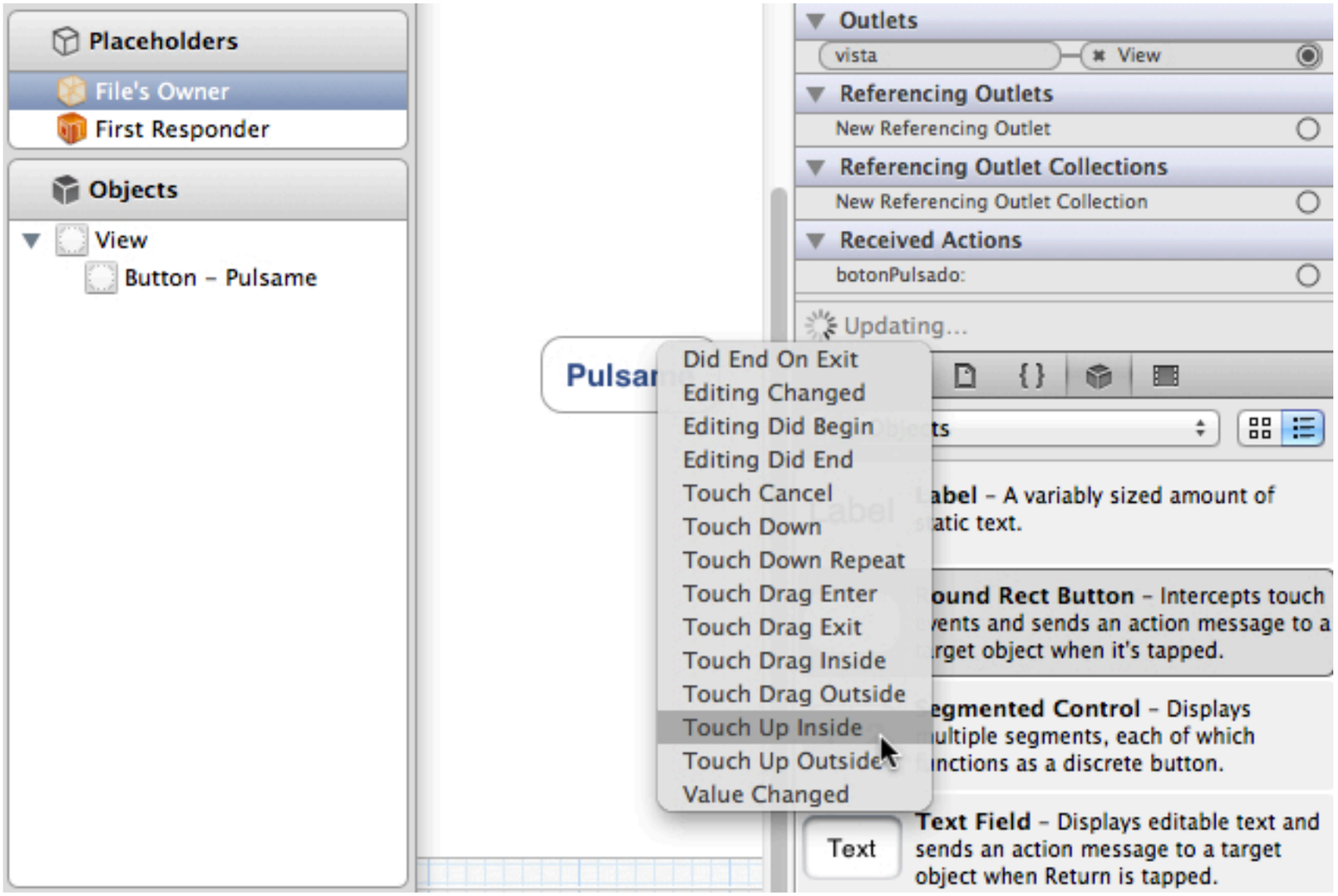
- Podemos conectar eventos con observadores de forma visual
- El método observador se declara como `IBAction`
  - En preprecesamiento se sustituye por `(void)`

```
- (IBAction)botonPulsado: (id)sender;
```

- Normalmente toma como parámetro el objeto que produjo el evento

Conexión con la acción

- Debemos conectar con un evento del objeto
  - Por ejemplo, *Touch Up Inside* en un botón



Creación programática de ventanas

- Crear la ventana que ocupe toda la pantalla (`UIScreen`)

```
UIWindow *window = [[UIWindow alloc]
        initWithFrame: [[UIScreen mainScreen] bounds]];
```

- Mostrar ventana en pantalla y establecerla como ventana clave

```
[self.window makeKeyAndVisible];
```

- Podemos obtener la ventana clave desde cualquier sitio

```
UIWindow *window  = [[UIApplication sharedApplication] keyWindow];
```

Creación programática de vistas

- Se inicializan con el rectángulo que abarcan en pantalla

```
UIView *vista = [[UIView alloc]
        initWithFrame:CGRectMake(0,0,100,100)];
```

- Rectángulo de la pantalla completa (sin barra de estado)

```
CGRect marcoVista = [[UIScreen mainScreen] applicationFrame];
```

- Añadir subvistas

```
[self.window addSubview: vista];
```

  - Podemos acceder a la supervista con la propiedad `superview`
- Buscar vista hija a partir de etiqueta

```
UIView *texto = [self.window viewWithTag: 1];
```

  - La etiqueta puede establecerse desde el inspector de atributos

Programación de eventos

- Podemos definir observadores de forma programática
  - Especificar *target* y *selector*

```
[boton addTarget: self action: @selector(botonPulsado:)
    forControlEvents: UIControlEventTouchUpInside];
```

  - Si como *target* ponemos `nil` busca en la jerarquía de clases la primera que defina el selector especificado
    Es el significado de *First Responder* en Interface Builder

- El selector puede tomar ninguno, uno, o dos parámetros

```
- (void)botonPulsado: (id) sender event: (UIEvent *) evento {
    ...
}
```

Propiedades de las vistas

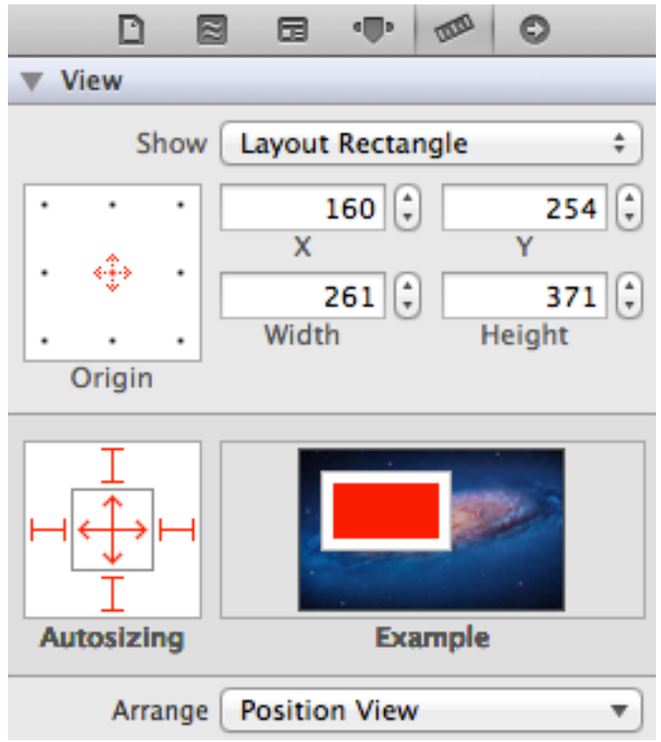
- Posición y límites de la vista
  - Configurable también desde el inspector de tamaño

```
// Limites en coordenadas locales. Su origen siempre es (0,0)
CGRect area = [vista bounds];
// Posición del centro de la vista en coordenadas de su supervista
CGPoint centro = [vista center];
// Marco en coordenadas de la supervista
CGRect marco = [vista frame]
```

- Transformación
  - Propiedad `transform`, por defecto `CGAffineTransformIdentity`
  - Macros `CGAffineTransformMakeRotation` y `CGAffineTransformMakeScale`
- Color de fondo y transparencia
  - Configurable también desde el inspector de atributos
  - Propiedad `backgroundColor` (`UIColor`)
  - Propiedades `alpha` (`CGFloat`) y `hidden` (`BOOL`)

Disposición

- Se puede configurar desde el inspector de tamaño



- Autoredimensionado
  - Los componentes se adaptan al tamaño de la ventana
  - Orientación horizontal / vertical
  - Se debe activar *Autoresize subviews*

    Propiedad `autoresizedSubviews`
  - Se configura en *Autosizing*

    O con la propiedad `autoresizingMask`

Controles básicos

- Los controles derivan de `UIControl` (ésta deriva de `UIView`)
- Añaden una serie de eventos (patrón observador)

- Campos de texto
  - Se implementan en `UITextField`
  - Se suelen establecer las propiedades *Text* y *Placeholder Text*
  - Se suele observar el evento *Value Changed*

- Botones
  - Se implementan en `UIButton`
  - Se suelen establecer las propiedades *Title* y *Text*
  - Se suele observar el evento *Touch Up Inside*

Imágenes

- Se representan mediante la clase `UIImage`
- Se suelen cargar del *bundle* principal

```
UIImage *imagen = [UIImage imageNamed: @"imagen.png"];
```

- Se pueden mostrar con `UIImageView`

```
UIImageView *imageView = [[UIImageView alloc] initWithImage: imagen];
```

- Soporte para pantalla retina (960 x 640)
  - Debemos incluir una versión de la imagen a doble resolución
  - Añadimos sufijo `@2x` al nombre del fichero

    `imagen.png` `imagen@2x.png`
  - Siempre cargamos la imagen con el nombre original

¿Preguntas...?