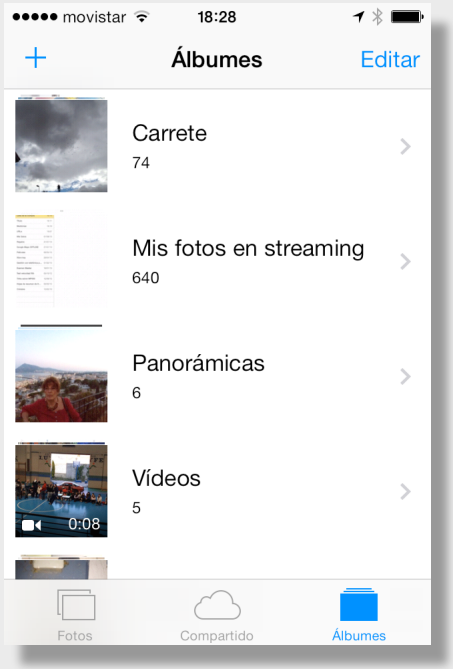
Desarrollo de Apps para iOS
Table Views

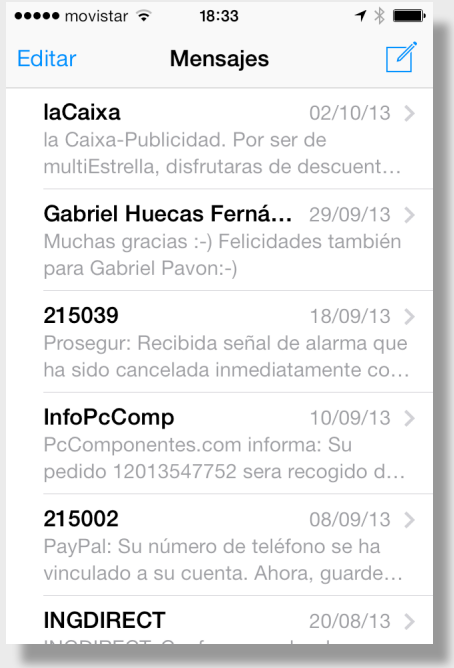
IWEB,LSWC 2013-2014
Santiago Pavón

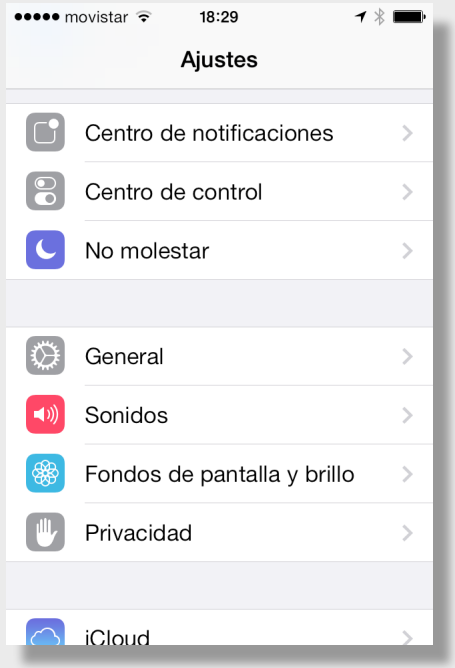
ver: 2014.03.23

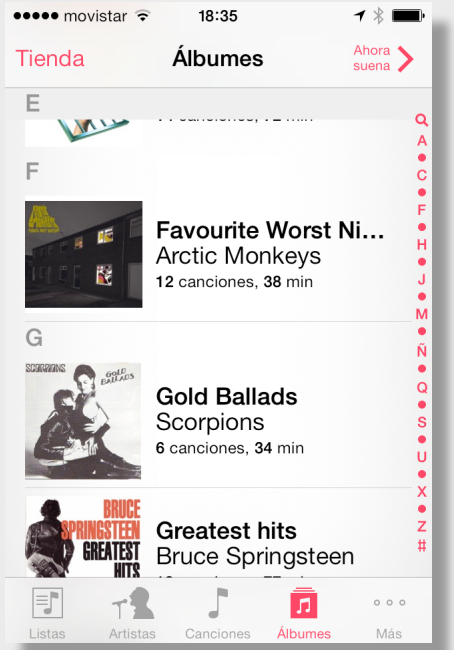
Características de las Tablas

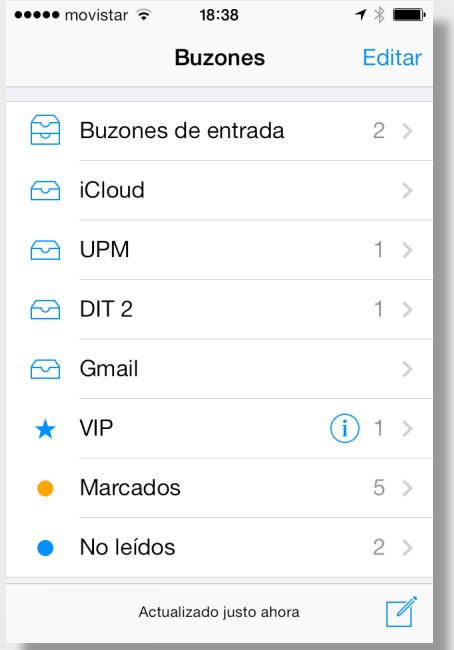
- Mostrar la información en una tabla de una columna.
  - Cada fila es una celda.
- Apariencia:
  - Pueden tener una cabecera y pie de tabla.
  - Se pueden agrupar las celdas en secciones.
    - Cada sección: cabecera, celdas y pie.
  - Existen varios estilos predefinidos de celdas, o podemos crear celdas personalizadas.
- Las tablas y celdas con objetos `UITableView` y `UITableViewCell`.
  - **UITableView** deriva de **UIScrollView**.
  - Uso muy eficiente de las celdas reutilizando las celdas no visibles.
- Las tablas pueden ser estáticas o dinámicas.
- Protocolos:
  - Fuente de datos: `UITableViewDataSource`.
  - Apariencia y comportamiento: `UITableViewDelegate`.











iOS 7

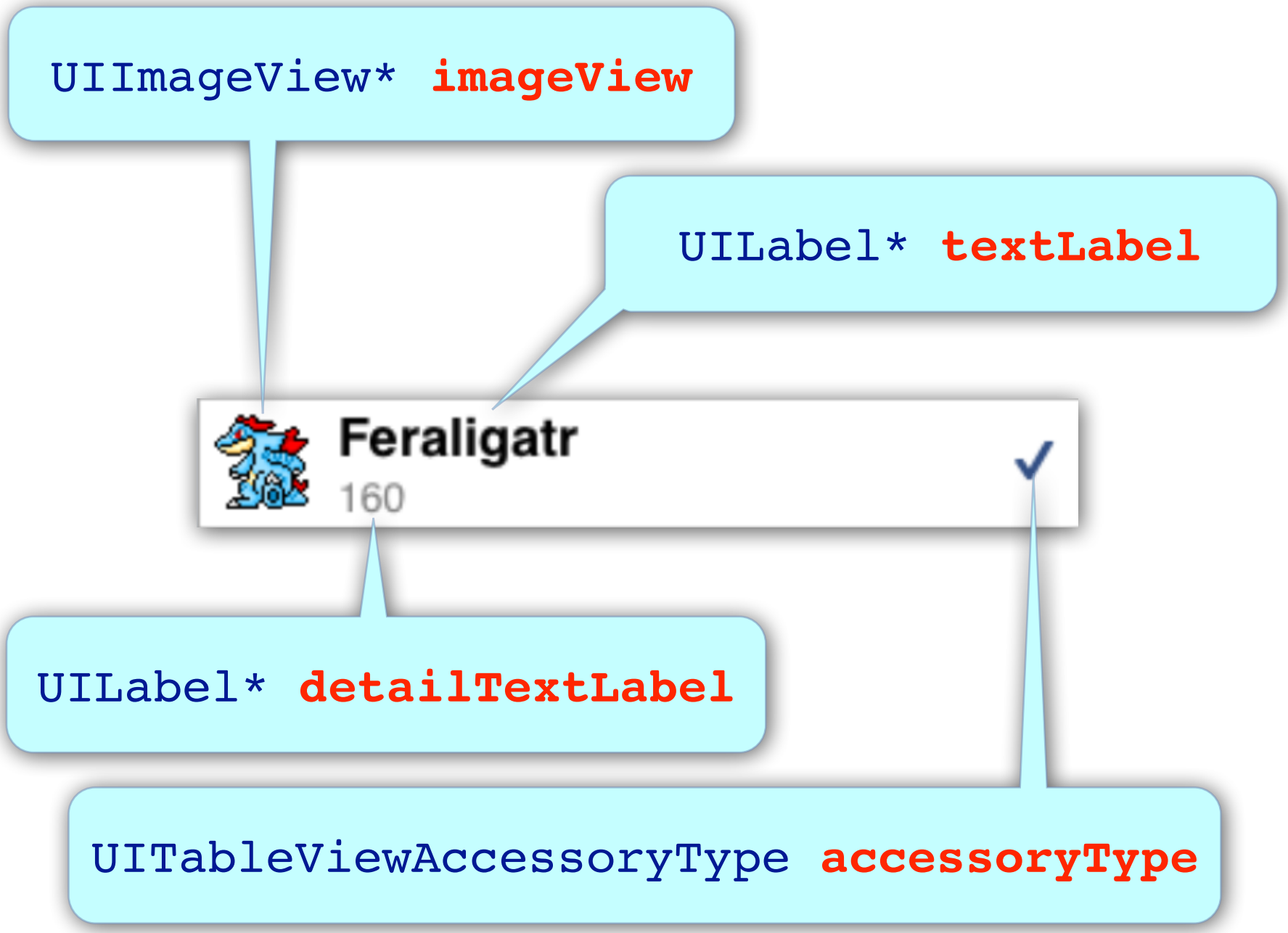
Estilos de Celdas Predefinidos

- **Basic** (**UITableViewCellStyleDefault**)

- **Subtitle** (**UITableViewCellStyleSubtitle**)

- **Right Detail** (**UITableViewCellStyleValue1**)

- **Left Detail** (**UITableViewCellStyleValue2**)



Propiedades de las celdas

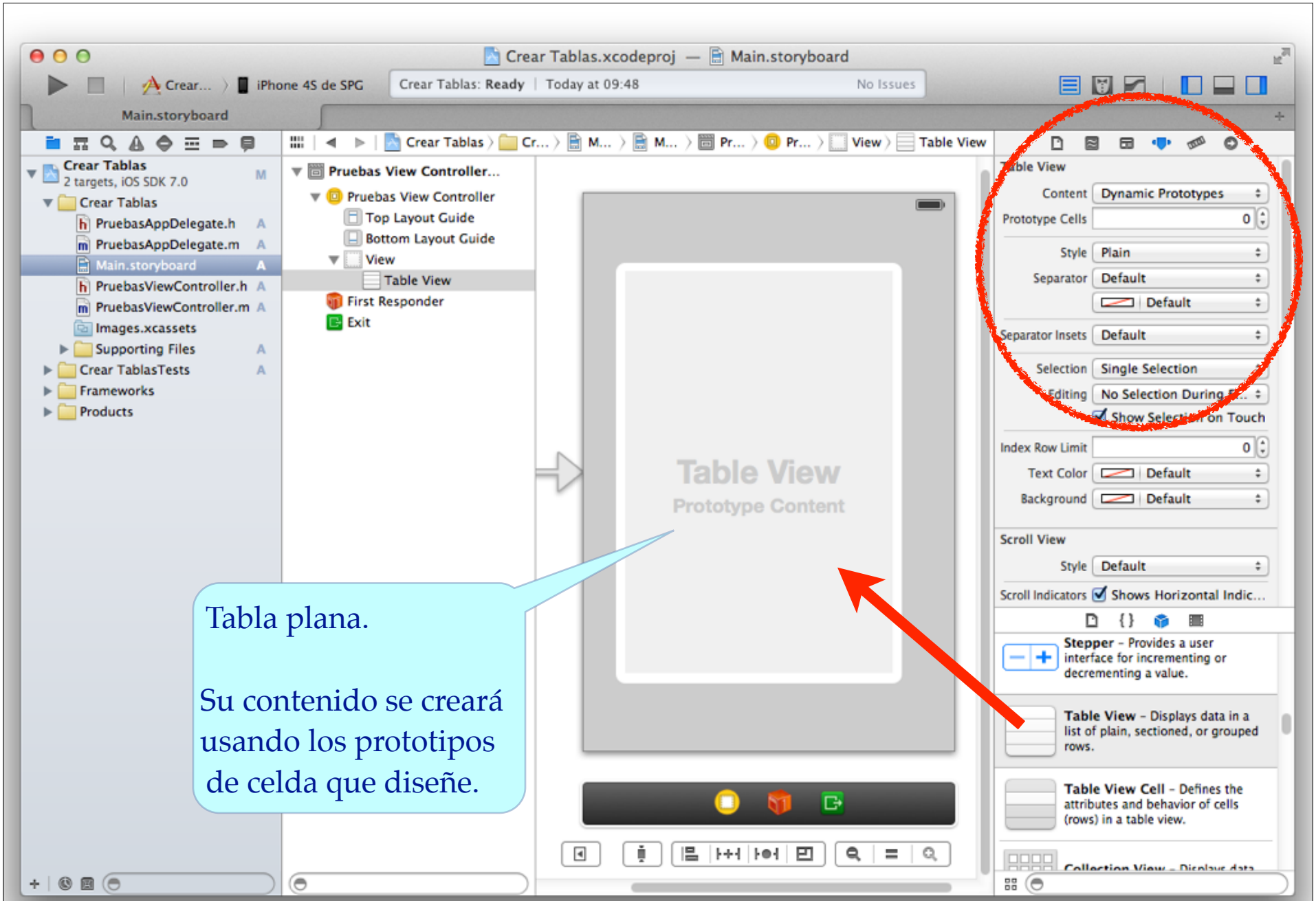
- **`imageView`**
  - Es una **`UIImageView`** que podemos personalizar cambiando sus propiedades
    - **`image`**, **`highlightedImage`**, . . .
- **`textLabel`**
  - Es una **`UILabel`** que podemos personalizar cambiando sus propiedades
    - **`text`**, **`font`**, . . .
- **`detailTextLabel`**
  - Es una **`UILabel`** que podemos personalizar cambiando sus propiedades
    - **`text`**, **`font`**, . . .
- **`contentView`**
  - Es la **`UIView`** donde se muestra el contenido de la celda.
  - Podemos añadir nuestras propias subviews para personalizar las celdas.
- **`accessoryType`**
  - Es el tipo de accesorio a mostrar en la celda.
- . . .

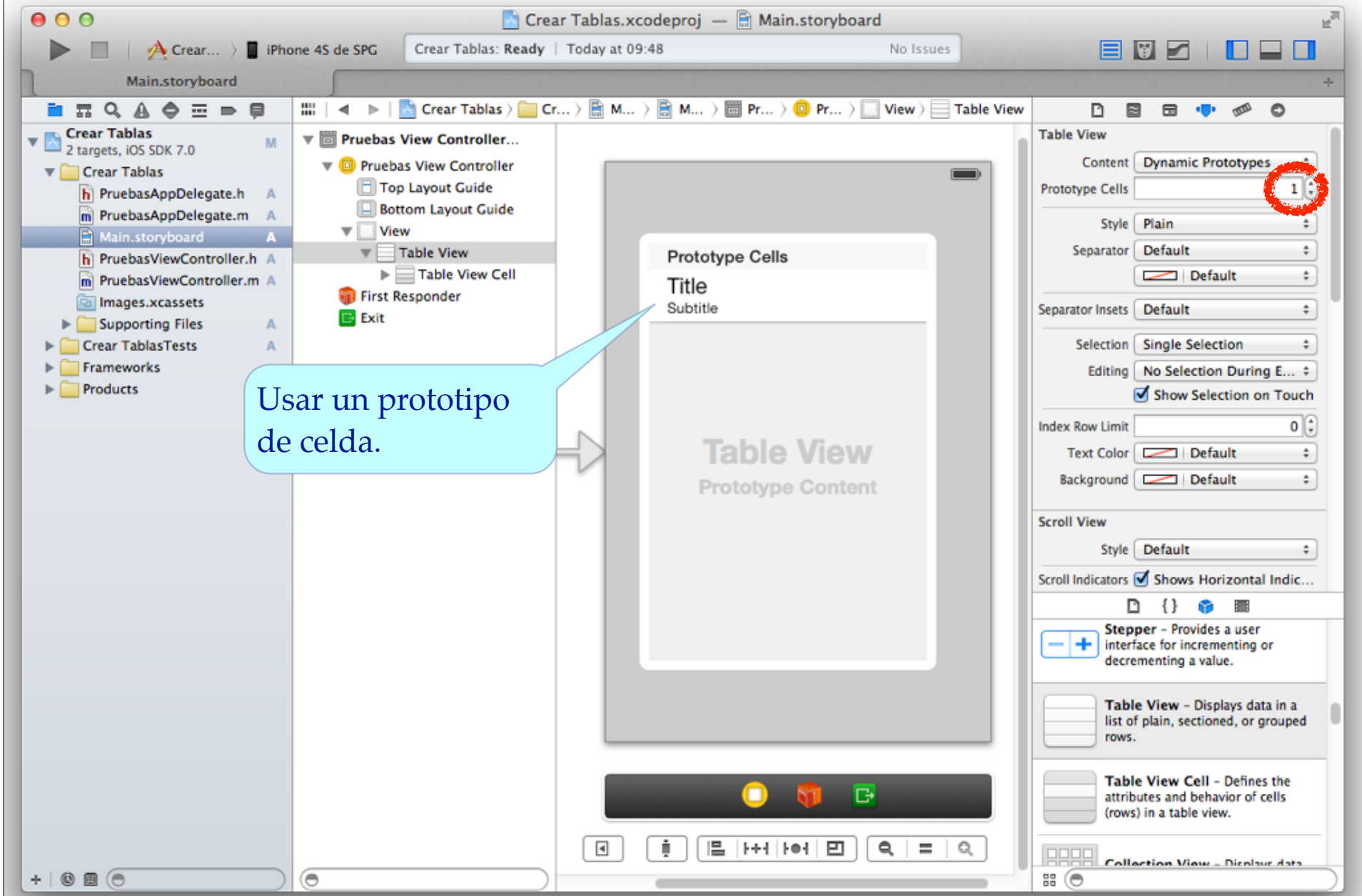
Crear un objeto UITableView

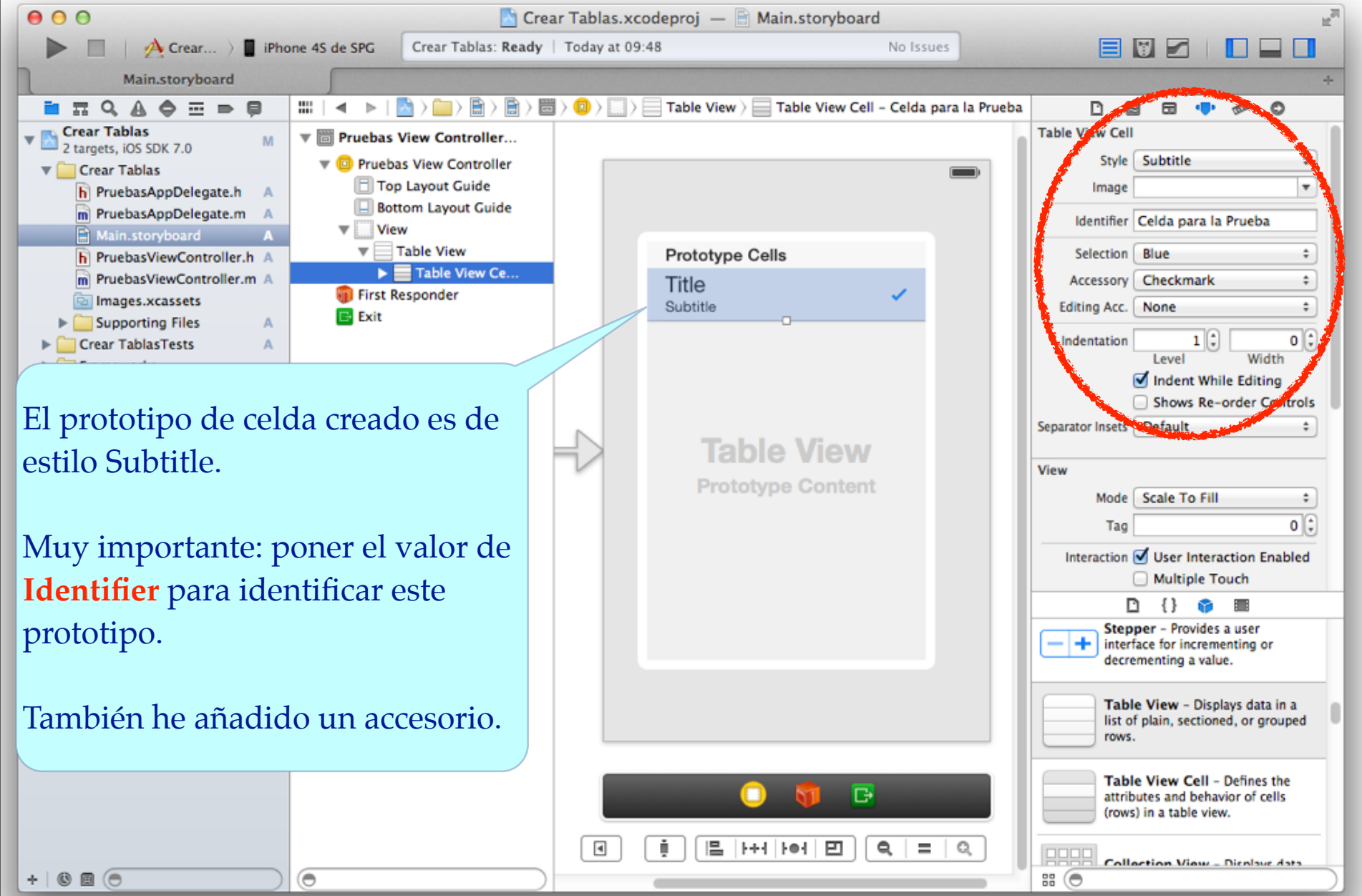
- **Programáticamente:**

```
CGRect rect = CGRectMake(50, 50, 220, 300);
UITableView * tv = [[UITableView alloc]
                                    initWithFrame:rect
                                    style:UITableViewStylePlain];
[self.view addSubview:tv];
```

- **Interface Builder y Storyboard:**
  - Arrastrar un objeto Table View desde la librería de objetos.
    - Configurar la tabla en el inspector.
      - Contenido: celdas creadas usando prototipos.
        - Las celdas estáticas solo en UITableViewController.
      - Número de prototipos de celdas que queremos usar.
      - Estilo de la tabla: plana, agrupada.
      - ...
    - Asignar su objeto delegado y data source.
    - ...



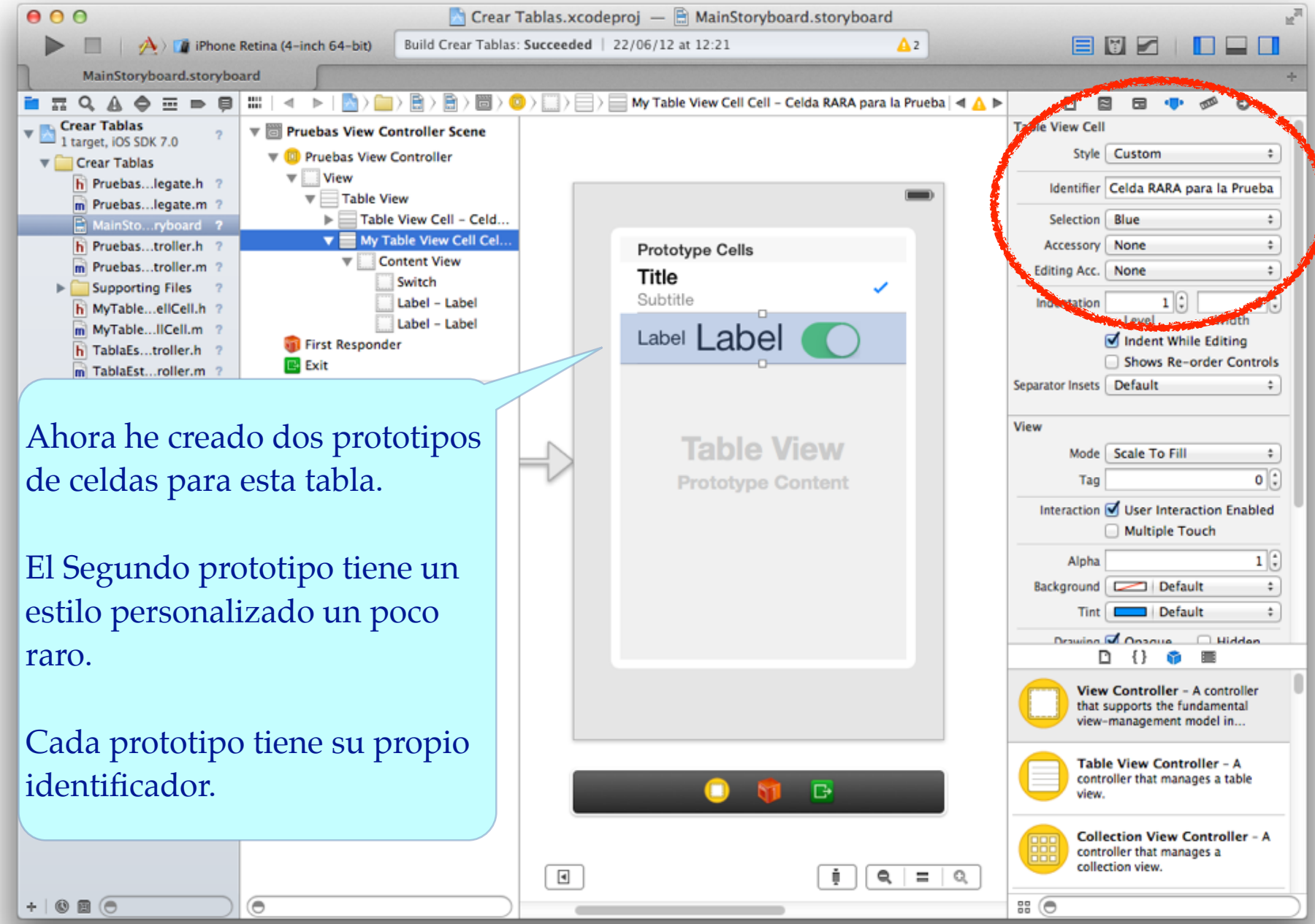


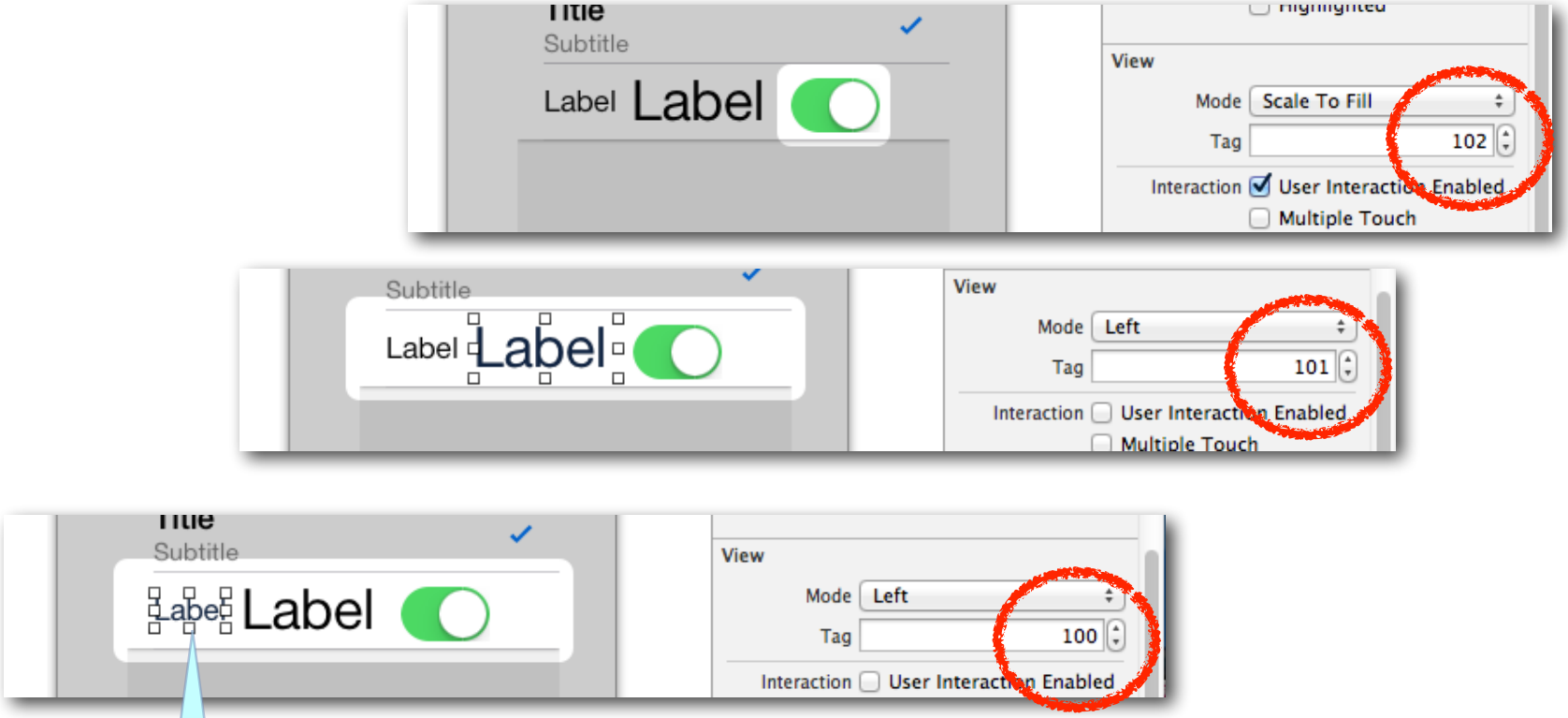


El prototipo de celda creado es de estilo Subtitle.

Muy importante: poner el valor de **Identifier** para identificar este prototipo.

También he añadido un accesorio.

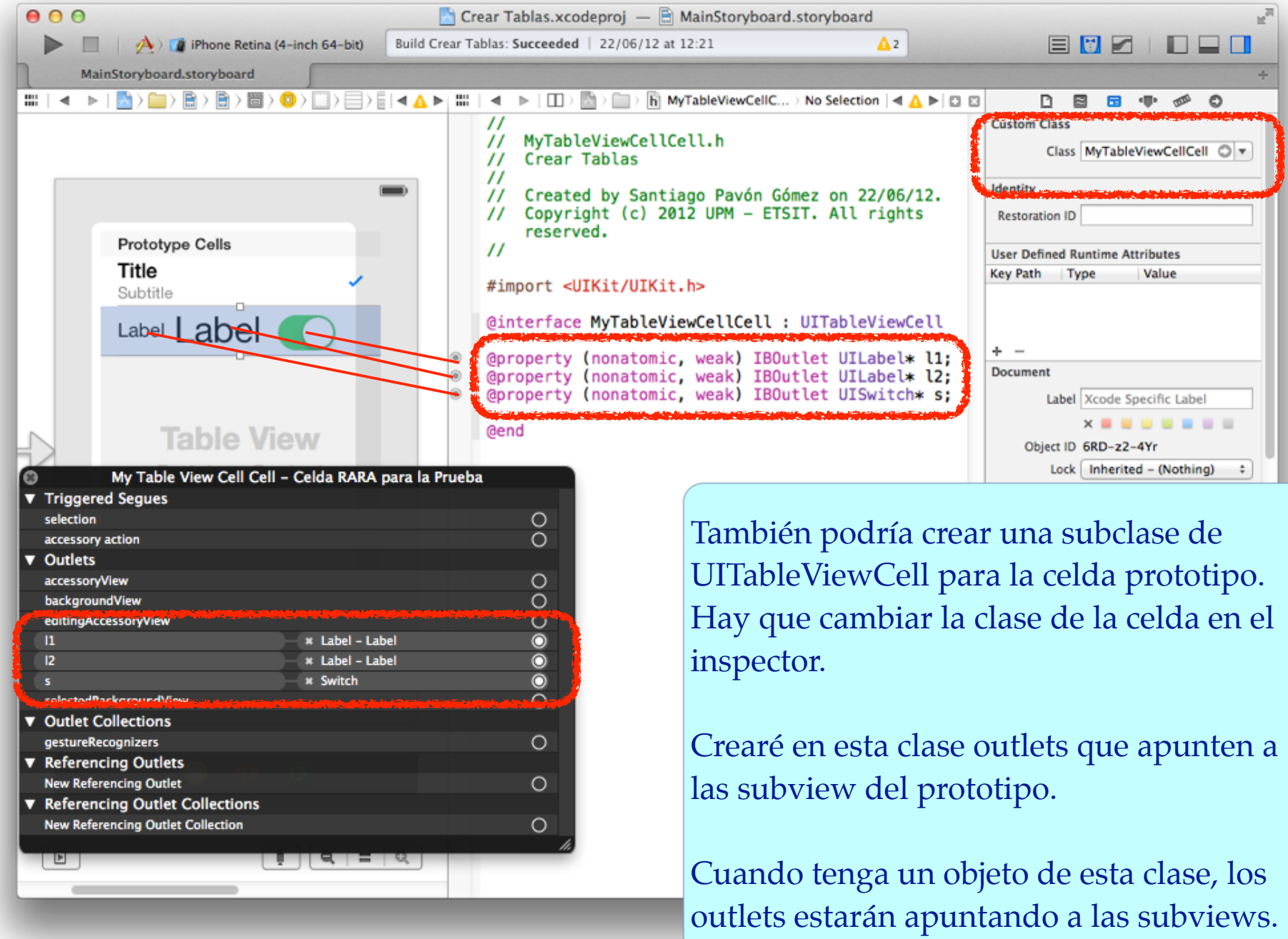




Para acceder a las subviews añadidas en este prototipo de celda personalizado, asigno un **tag** distinto a cada subview.

Cuando tenga una celda, accederé a la subview que desee usando el método **-viewWithTag:**

```
UILabel * l = (UILabel*)[cell viewWithTag:100];
```



También podría crear una subclase de UITableViewCell para la celda prototipo. Hay que cambiar la clase de la celda en el inspector.

Crearé en esta clase outlets que apunten a las subview del prototipo.

Cuando tenga un objeto de esta clase, los outlets estarán apuntando a las subviews.

Obtener los Datos a Mostrar

- La Table View **no posee** los datos.
  - Se los proporciona otro objeto que actúa como Data Source.
- Usa el protocolo `UITableViewDataSource` para obtener los datos.
  - Las tablas tienen una propiedad llamada `dataSource` que debe apuntar a un objeto conforme a este protocolo.
  - El objeto dataSource **no se retiene**.
    - Se está suponiendo que el objeto data source tendrá una vida superior a la de la tabla.
  - La tabla le pregunta a su dataSource:
    - cuántas secciones hay.
    - cuántas filas hay en cada sección.
    - y le pide que le proporcione las celdas a colocar en cada fila de cada sección.

UITableViewDataSource

- **Configurar la Table View**
  - **`tableView:cellForRowAtIndexPath:`** (método requerido)
  - `numberOfSectionsInTableView:`
  - **`tableView:numberOfRowsInSection:`** (método requerido)
  - `sectionIndexTitlesForTableView:`
  - `tableView:sectionForSectionIndexTitle:atIndex:`
  - `tableView:titleForHeaderInSection:`
  - `tableView:titleForFooterInSection:`
- **Añadir o borrar filas**
  - `tableView:commitEditingStyle:forRowAtIndexPath:`
  - `tableView:canEditRowAtIndexPath:`
- **Reordenar las filas**
  - `tableView:canMoveRowAtIndexPath:`
  - `tableView:moveRowAtIndexPath:toIndexPath:`

NSIndexPath

- Los objetos de la clase **`NSIndexPath`** identifican la posición de las celdas dentro de una tabla.
  - Es el número de sección y fila de una celda de la tabla.
- Tiene dos propiedades:
  - **`row`** y **`section`**.
    - devuelven un entero (**`NSInteger`**).

Reutilización de celdas

- Crear y destruir celdas es costoso.
  - Las celdas no visibles, se guardan en una cola para reutilizarlas.
- Si se necesita una celda nueva, se sacar de la cola de reutilización con:

```
- (UITableViewCell*)
      dequeueReusableCellWithIdentifier:(NSString*)identifier;
                            forIndexPath:(NSIndexPath*)indexPath
```

  - Si la cola está vacía se construye automáticamente una nueva celda copiando el prototipo.
    - Si la cola está vacía y no hubiéramos definido un prototipo, se devuelve **nil**, y hay que crear la nueva celda programáticamente.

```
-(id) initWithStyle:(UITableViewCellStyle)style
    reuseIdentifier:(NSString*)reuseId;
```

```
- (UITableViewCell *)tableView:(UITableView *)tableView
        cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *myId = @"MiCeldaId";

    UITableViewCell *cell =
             [tableView dequeueReusableCellWithIdentifier:myId
                                             forIndexPath:indexPath];

    if (cell == nil) {
        cell = [[UITableViewCell alloc]
                      initWithStyle:UITableViewCellStyleDefault
                    reuseIdentifier:myId];
    }

    NSUInteger row     = indexPath.row;
    NSUInteger section = indexPath.section;

    cell.imageView.image = [self.almacen getPhoto:row];
    cell.imageView.highlightedImage = [self.almacen getHPhoto:row];
    cell.textLabel.text = [self.almacen getName:row];
    cell.accessoryType = UITableViewCellAccessoryCheckmark;

    return cell;
}
```

Identificar tipo de celda o prototipo

Usando prototipos no hace falta la sentencia **if**.

UITableView - Actualizaciones

```
-(void) reloadData;

-(void) insertSections:(NSIndexSet *)sections
      withRowAnimation:(UITableViewRowAnimation)animation;

-(void) deleteSections:(NSIndexSet *)sections
      withRowAnimation:(UITableViewRowAnimation)animation;

-(void) reloadSections:(NSIndexSet *)sections
      withRowAnimation:(UITableViewRowAnimation)animation;

-(void) insertRowsAtIndexPaths:(NSArray *)indexPath
             withRowAnimation:(UITableViewRowAnimation)animation;

-(void) deleteRowsAtIndexPaths:(NSArray *)indexPaths
             withRowAnimation:(UITableViewRowAnimation)animation;

-(void) reloadRowsAtIndexPaths:(NSArray *)indexPaths
             withRowAnimation:(UITableViewRowAnimation)animation;
```

El delegado de la tabla

- Los objetos **`UITableView`** tienen una propiedad llamada **`delegate`**.
  - El objeto delegado **no se retiene**.
    - Se está suponiendo que el objeto delegado tendrá una vida superior a la de la tabla.
- Debe apuntar a un objeto que sea conforme con el protocolo **`UITableViewDelegate`**.
- El objeto delegado maneja la selección de celdas, ayuda en el borrado y la reordenación de las celdas, configura las cabeceras y pies de las secciones, observa las acciones realizadas sobre la tabla, controla cómo se muestra la tabla, ...

UITableViewDelegate

- Configurar las filas de la tabla:
  - `tableView:heightForRowAtIndexPath:`
  - `tableView:estimatedHeightForRowAtIndexPath:`
  - `tableView:indentationLevelForRowAtIndexPath:`
  - `tableView:willDisplayCell:forRowAtIndexPath:`
- Manejar los accesorios:
  - `tableView:accessoryButtonTappedForRowWithIndexPath:`
- Manejar la selección de celdas:
  - `tableView:willSelectRowAtIndexPath:`
  - `tableView:didSelectRowAtIndexPath:`
  - `tableView:willDeselectRowAtIndexPath:`
  - `tableView:didDeselectRowAtIndexPath:`

- **Modificar la cabecera y pie de las secciones:**
  - `tableView:viewForHeaderInSection:`
  - `tableView:viewForFooterInSection:`
  - `tableView:heightForHeaderInSection:`
  - `tableView:estimatedHeightForHeaderInSection:`
  - `tableView:heightForFooterInSection:`
  - `tableView:estimatedHeightForFooterInSection:`
  - `tableView:willDisplayHeaderView:forSection:`
  - `tableView:willDisplayFooterView:forSection:`
- **Edición de las filas de la tabla:**
  - `tableView:willBeginEditingRowAtIndexPath:`
  - `tableView:didEndEditingRowAtIndexPath:`
  - `tableView:editingStyleForRowAtIndexPath:`
  - `tableView:titleForDeleteConfirmationButtonForRowAtIndexPath:`
  - `tableView:shouldIndentWhileEditingRowAtIndexPath:`
- **Reordenar las filas de la tabla:**
  - `tableView:targetIndexPathForMoveFromRowAtIndexPath:`
    `toProposedIndexPath:`

- Seguimiento del borrado de views:
  - `tableView:didEndDisplayingCell:forRowAtIndexPath:`
  - `tableView:didEndDisplayingHeaderView:forSection:`
  - `tableView:didEndDisplayingFooterView:forSection:`

- Gestionar el copiado y pegado del contenido de las filas:
  - `tableView:shouldShowMenuForRowAtIndexPath:`
  - `tableView:canPerformAction:forRowAtIndexPath:withSender:`
  - `tableView:performAction:forRowAtIndexPath:withSender:`

- Gestionar el resaltado de las filas:
  - `tableView:shouldHighlightRowAtIndexPath:`
  - `tableView:didHighlightRowAtIndexPath:`
  - `tableView:didUnhighlightRowAtIndexPath:`

Al Seleccionar una Fila ...

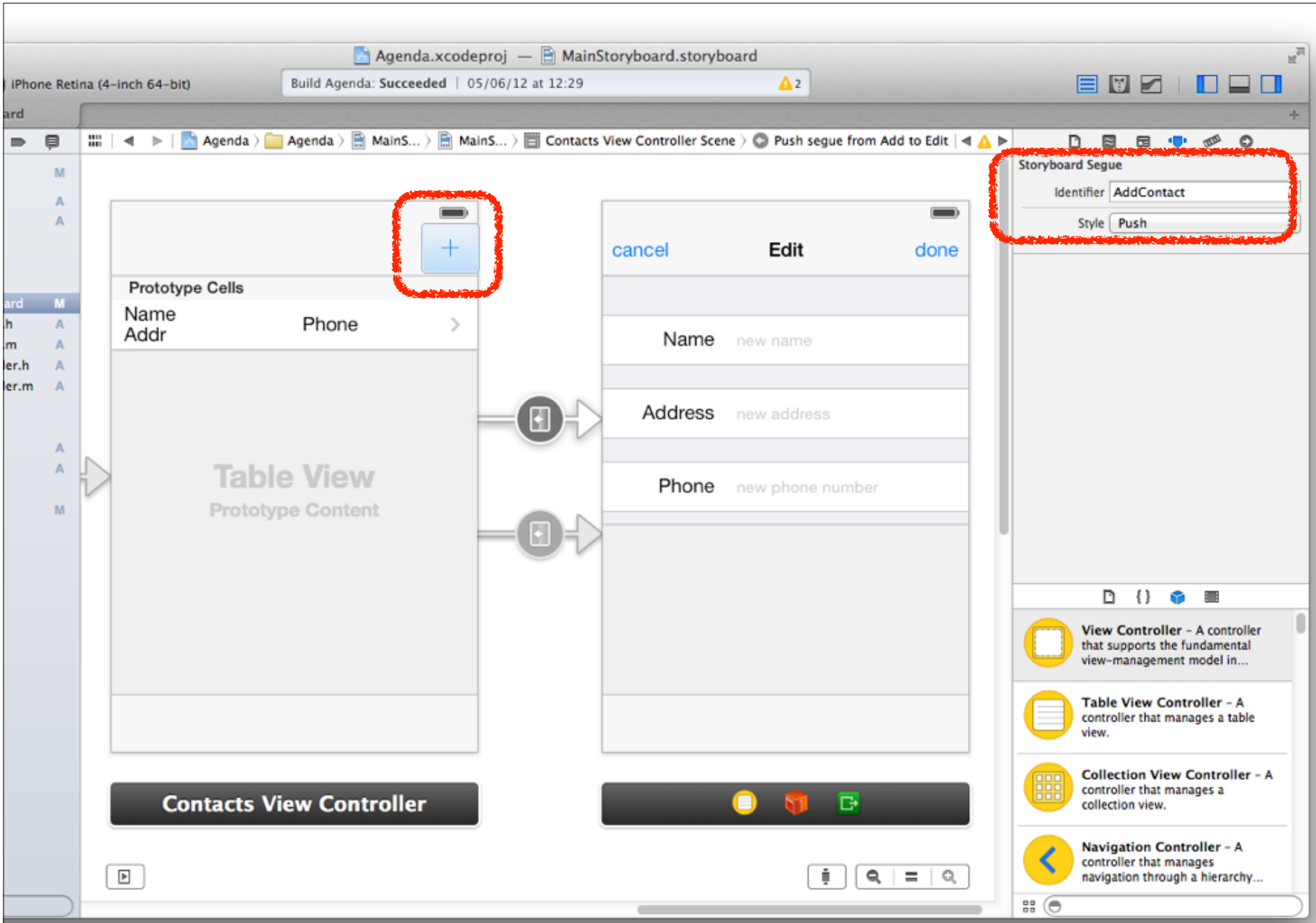
- Cuando (des)seleccionamos o vamos a (des)seleccionar una fila de la tabla se invoca un método del delegado.
  - Por ejemplo, al seleccionar una fila se invoca:
    - **`tableView:didSelectRowAtIndexPath:`**
      - Nos pasan el index path de la fila seleccionada.
      - Este método se usa típicamente para programar alguna acción, por ejemplo, mostrar modalmente otro VC, navegar a otra pantalla.
- También podemos crear un segue que se dispare cuando se selecciona una fila de la tabla.
  - Asignaremos un identificador al segue.
  - Adaptaremos el método **`prepareForSegue:sender:`** para:
    - Consultar que celda está seleccionada para saber dónde hemos pulsado.
    - Configurar el VC destino del segue.

Ejemplo

```
- (void)          tableView:(UITableView*)tableView
    didSelectRowAtIndexPath:(NSIndexPath*)indexPath
{

    // Creo un objeto VC nuevo cargandolo del storyboard
    UIStoryboard * storyboard = self.storyboard;
    InfoViewController *ivc = [storyboard
          instantiateViewControllerWithIdentifier:@"Info VC"];

    // Configuro un parametro del VC creado
    ivc.dato = indexPath.row;

    // Paso el VC al Navigation Controller para que lo muestre
    [self.navigationController pushViewController:ivc
                                         animated:YES];
}
```

Ejemplo

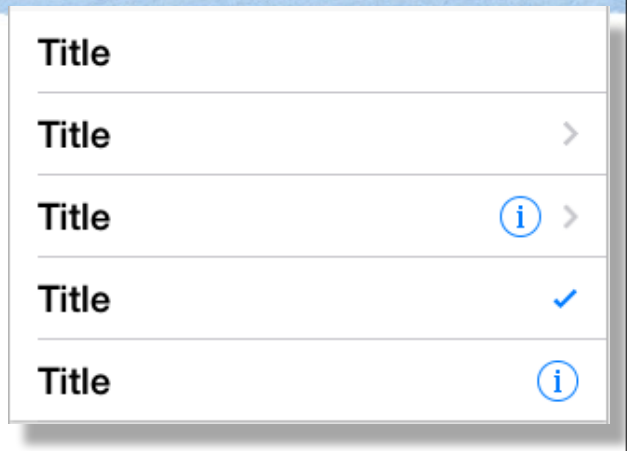
```
- (void) prepareForSegue:(UIStoryboardSegue *)segue sender:(id)sender {
    if ([segue.identifier isEqualToString:@"EditContact"]) {
        EditContactViewController * ecvc = segue.destinationViewController;
        NSInteger row = [self.tableView indexPathForCell:sender].row;
        ecvc.contact = contactsTable[row];
    } else if ([segue.identifier isEqualToString:@"AddContact"]) {

        EditContactViewController * ecvc = segue.destinationViewController;
        Contact * newContact = [[Contact alloc] initWithName:@""
                                                     address:@""
                                                     phone:@""];
        [contactsTable insertObject:newContact atIndex:0];
        NSIndexPath * ip = [NSIndexPath indexPathForRow:0 inSection:0];
        [self.tableView insertRowsAtIndexPaths:@[ip]
                              withRowAnimation:YES];
        ecvc.contact = newContact;
    }
}
```



Accesorios

- Controles accesorios usados por las celdas

```
UITableViewCellAccessoryNone
UITableViewCellAccessoryDisclosureIndicator
UITableViewCellAccessoryDetailDisclosureButton
UITableViewCellAccessoryCheckmark
UITableViewCellAccessoryDetailButton
```



- Cuando el usuario pulsa en el accesorio **Detail Disclosure Button** o **Detail Button**, se ejecuta en el delegado el método:

```
–tableView:accessoryButtonTappedForRowWithIndexPath:
```

- Desde los accesorios de tipo **Detail Disclosure Button** y **Detail Button** de las celdas también se pueden lanzar segues.
  - La celda con el accesorio hace el papel de sender del segue.

Editar una Tabla

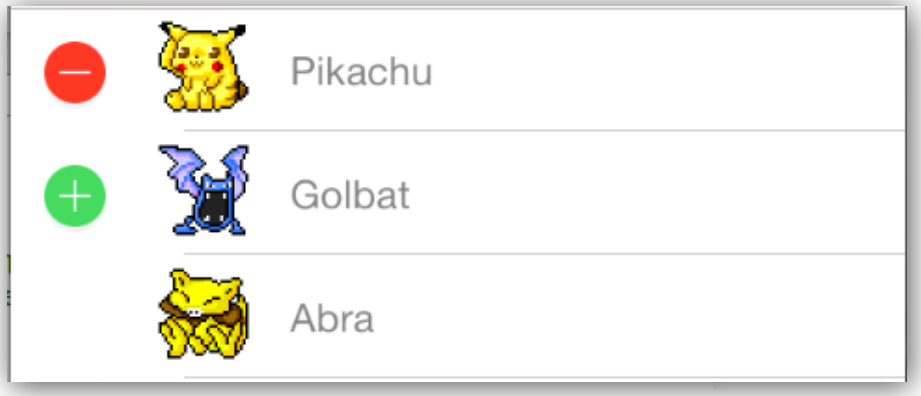
- Preguntar al **data source** si la celda es editable.

```
- (BOOL) tableView:(UITableView*)
         canEditRowAtIndexPath:(NSIndexPath*)
```

- Preguntar al **delegado** por el estilo de edición:
  - borrar la celda, insertar una celda nueva, o no editar.

```
- (UITableViewCellEditingStyle)tableView:(UITableView*)
                editingStyleForRowAtIndexPath:(NSIndexPath*)
```

  - Se muestra un icono indicando el tipo de edición



- Cuando el usuario toca alguno de los botones de pa pantalla para borrar una fila o insertar una fila en la tabla, se llama automáticamente al método del **data source**:

```
- (void)   tableView:(UITableView*)
  commitEditingStyle:(UITableViewCellEditingStyle)
   forRowAtIndexPath:(NSIndexPath*)
```

  - Este método del data source debe encargarse de:
    - Cambiar los datos almacenados en el modelo.
    - Actualizar la UITableView llamando a alguno de sus métodos de actualización/refresco:

```
-reloadData
-insertRowsAtIndexPaths:withRowAnimation:
-deleteRowsAtIndexPaths:withRowAnimation:
```

- También podemos reordenar las celdas usando los controles de las Table Views:
  - Los métodos y propiedades relacionados con la reordenación de celdas que deberemos manejar son:
    - Preguntar al data source si una celda puede moverse a otra posición:

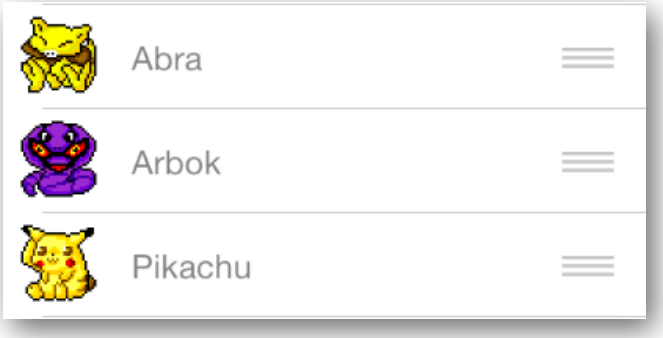
```
- (BOOL)      tableView:(UITableView*)
  canMoveRowAtIndexPath:(NSIndexPath*)
```

    - Decirle al data source que mueva una celda a otra posición:

```
- (void)  tableView:(UITableView*)
  moveRowAtIndexPath:(NSIndexPath*)
         toIndexPath:(NSIndexPath*)
```

    - Propiedad de la UITableViewCell que indica si debe mostrarse el control de reordenación en la celda:

```
@property BOOL showsReorderControl
```



UITableViewController

- Es una clase derivada de **`UIViewController`**
  - que tiene una **`UITableView`** como su view.
    - La propiedad **`view`** y **`tableView`** apuntan a esta tabla interna.
  - El propio objeto **`UITableViewController`** es el **data source** y **delegado** de su tabla interna.
- Proporciona facilidades:
  - carga inicial de datos
  - gestión del teclado al editar.
  - facilidades de edición.
  - etc.
- Con Interface Builder podemos añadir objetos **`UITableViewController`** al storyboard arrastrándolos desde la librería de objetos.
  - Crearemos para ellos subclases personalizadas que deriven de **`UITableViewController`**.
    - En el Inspector reasignaremos las clases.
- Y por supuesto, también se pueden crear programáticamente.

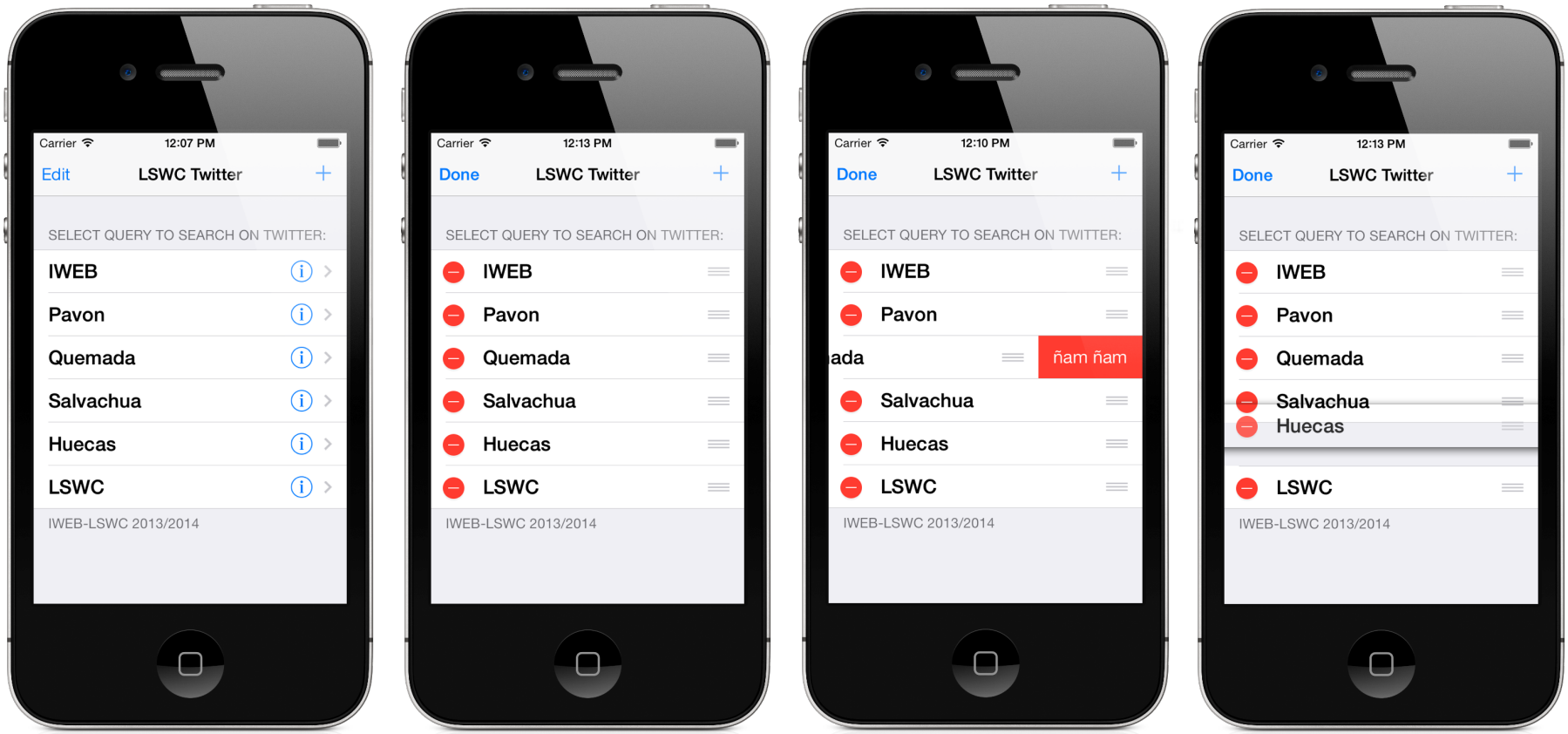
Editar una `UITableViewController`

- Los objetos **UIViewController** tienen una propiedad booleana (**`editing`**) que indican si estoy editando el contenido del VC.
- También proporcionan un método (**`-editButtonItem`**) que devuelve un botón que podemos colocar en la barra de navegación.
  - Este botón nos permite activar y desactivar el modo edición.
    - Según el estado cambia su título entre **Edit** y **Done**.
- Para cambiar el estado de la propiedad **`editing`** de forma animada podemos usar el método:

```
- (void)setEditing:(BOOL)editing
          animated:(BOOL)animated
```

- **UITableViewController usa internamente este estado (heredado de UIViewController) para gestionar la edición del contenido la tabla.**

Ejemplo: editar una tabla



```
- (void)viewDidLoad
{
    [super viewDidLoad];

    // display an Edit button in the navigation bar
    self.navigationItem.leftBarButtonItem = self.editButtonItem;

    [self loadQueries];
}


- (NSString *)tableView:(UITableView *)tableView
  titleForDeleteConfirmationButtonForRowAtIndexPath:(NSIndexPath *)ip
{
    return @"ñam ñam";
}


- (BOOL)tableView:(UITableView *)tableView
        shouldIndentWhileEditingRowAtIndexPath:(NSIndexPath*)ip
{
    return NO;
}
```

```
- (void)     tableView:(UITableView *)tableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
     forRowAtIndexPath:(NSIndexPath *)indexPath
{
    if (editingStyle == UITableViewCellEditingStyleDelete) {

        // Borro el dato de esa fila de mi modelo.
        [self.queries removeObjectAtIndex:indexPath.row];

        // Actualizar lo que muestra la table view
        [self.tableView deleteRowsAtIndexPaths:@[indexPath]
                              withRowAnimation:UITableViewRowAnimationAutomatic];

        [self saveQueries]; // persistencia

    } else if (editingStyle == UITableViewCellEditingStyleInsert) {

        // Create a new instance of the appropriate class,
        // insert it into the array,
        // and add a new row to the table view.
    }

}
```

```
- (void)          tableView:(UITableView *)tableView
       moveRowAtIndexPath:(NSIndexPath *)fromIndexPath
              toIndexPath:(NSIndexPath *)toIndexPath
{

    id obj = [self.queries objectAtIndex:fromIndexPath.row];

    [self.queries removeObjectAtIndex:fromIndexPath.row];
    [self.queries insertObject:obj atIndex:toIndexPath.row];

    [self saveQueries];
}


- (BOOL)           tableView:(UITableView *)tableView
      canMoveRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Return NO if you do not want the item to be re-orderable.
    return YES;
}
```

Celdas Prototipo y Estáticas

- Las Tablas pueden crearse para que usen
  - Celdas **prototipo**
    - las celdas de la tabla se crean copiando las celdas prototipo.
      - Cuando se necesitan más celdas, pueden reutilizarse celdas ya usadas y no visibles.
  - Celdas **estáticas** (***Solo para UITableViewController***)
    - La tabla sólo tiene las celdas estáticas que hemos creado con IB.
      - No se usan los prototipos para crear más celdas.
    - Cuando se usan celdas estáticas no tiene sentido usar el protocolo Data Source.
      - aunque podría usarse, evitando conflictos entre el diseño estático y la información proporcionada por el data source.
- Si el estilo de celda es personalizado (tanto para prototipos como para estáticas), para acceder a las subviews incluidas en una celda, podemos:
  - usar tags, asignando un tag distinto a cada subview.
  - crear una clase derivada de UITableViewCell y crear outlets apuntando a las subviews.

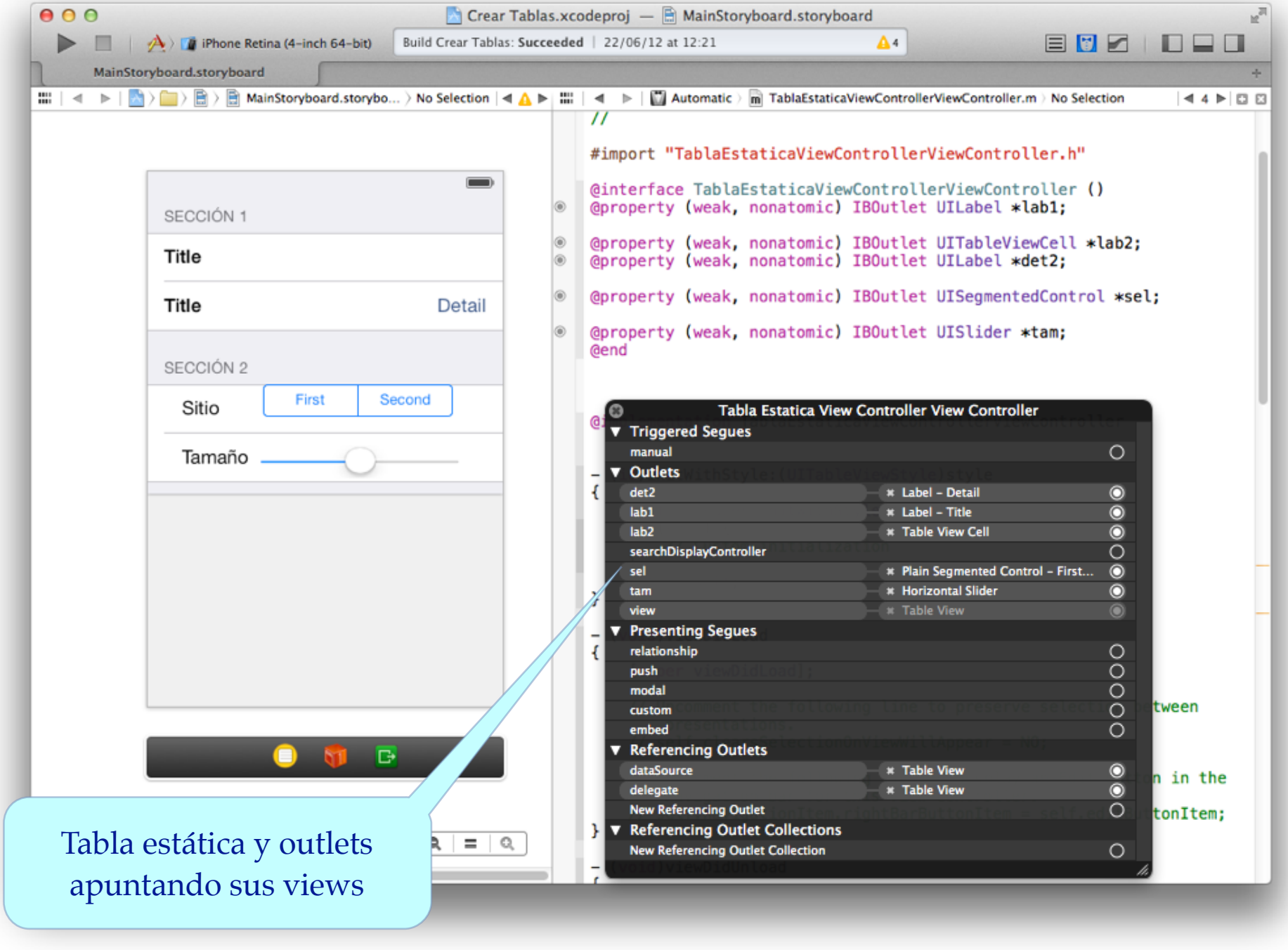


Tabla estática y outlets apuntando sus views

Altura de las Celdas

- El layout del contenido de las celdas puede hacerse con Springs y Structs, o con Auto Layout.
  - **Pero no se mira este layout para calcular cuál es la altura con la que deben pintarse las celdas.**
- La altura con la que pintan las celdas puede configurarse desde el inspector del **Interface Builder**.
  - Todas las celdas se pintan con la altura configurada, independientemente de cual sea la altura necesaria para pintar su contenido.
  - Si la altura del contenido de la celda es fijo y conocido, es muy fácil ajustar el tamaño de las celdas con Interface Builder para que se vea completamente.
- Si la altura del contenido de las celdas no es conocido apriori, o cada celda tiene una altura distinta, debemos sobreescribir el método **`-tableView:heightForRowAtIndexPath:`** para indicar que altura debemos usar para cada celda.
- Cuando se carga una tabla, este método se llama para todas las filas de la tabla.
  - Este cálculo inicial de alturas puede tardar mucho si la tabla tiene muchas celdas.
  - Para mejorar la experiencia de usuario al cargar tablas grandes, podemos estimar un valor para la altura de las celdas.
    - Estamos retrasando el cálculo de las alturas reales hasta el momento de hacer un scroll.
    - El valor estimado de altura se debe asignar a la propiedad **estimatedRowHeight** de las Table Views.
    - Para proporcionar una estimación individual para cada una de las celdas de la tabla podemos sobreescribir el método **-tableView:estimatedHeightForRowAtIndexPath:** del delegado de la tabla.

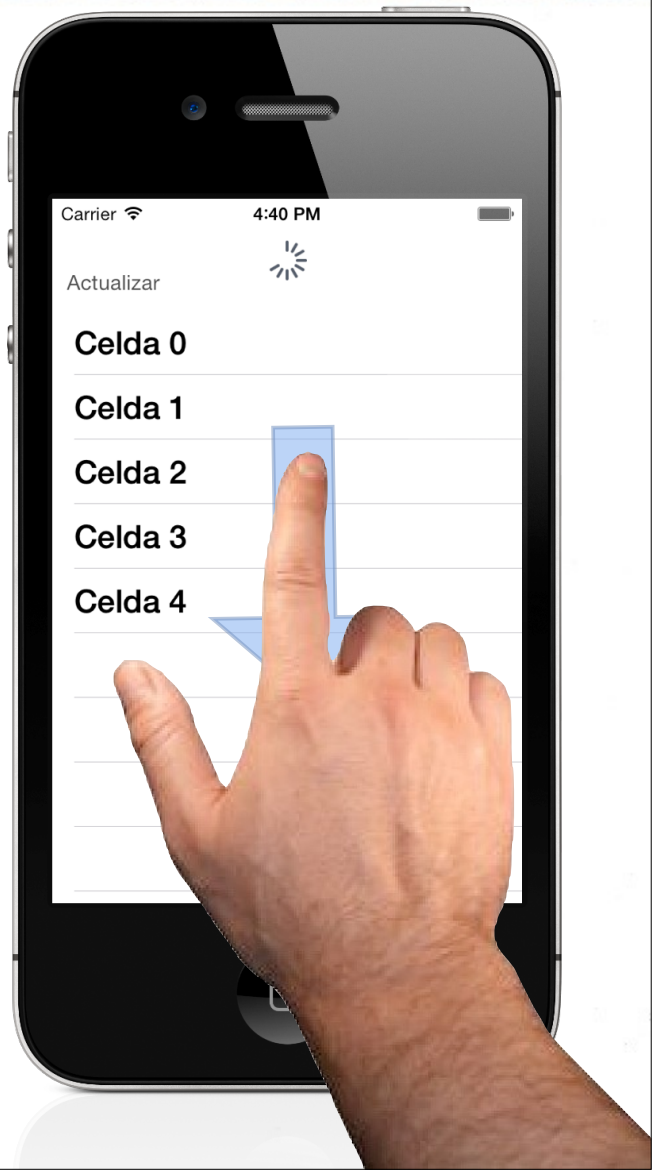
- Ejemplo: implementación del método de cálculo de la altura de las celdas de una tabla que solo muestra fotos:

```
- (CGFloat)          tableView:(UITableView *)tableView
      heightForRowAtIndexPath:(NSIndexPath *)indexPath {

    static UITableViewCell *cell;   // Uso siempre la misma celda

    if (cell == nil) {  // La primera vez creo una celda
       cell = [tableView dequeueReusableCellWithIdentifier:@"PhotoCell"];
    }

    // Configurar la celda: poner la imagen con la foto en la celda
    UIImageView *iv = (UIImageView *)[cell viewWithTag:100];
    iv.image = self.photos[indexPath.row];

    // Devolver la altura necesaria para que entre todo el contenido
    // cumpliendo las restricciones de autolayout
    return [cell.contentView systemLayoutSizeFittingSize:
                             UILayoutFittingCompressedSize].height + 1.0f;
}
```

Estirar para Refrescar

Estirar para Refrescar

- En las UITableViewController se puede añadir un control para refrescar / actualizar el contenido de la tablas al estirar hacia abajo desde el principio de una tabla



Programáticamente

```
- (void)viewDidLoad {
    [super viewDidLoad];

    UIRefreshControl *refreshControl = [[UIRefreshControl alloc] init];

    [refreshControl addTarget:self
                       action:@selector(refreshControlValueChanged)
             forControlEvents:UIControlEventValueChanged];

    self.refreshControl = refreshControl;
}

-   (void)refreshControlValueChanged {
    // Descargo los nuevos datos
    . . .

    // Recargar tabla
    [self.tableView reloadData];

    // Terminar el control de refresco
    [self.refreshControl endRefreshing];
}
```

Creo un objeto.

Evento que se genera al estirar.

Guardo el objeto en la propiedad de la UITableViewController

Indicar que el refresco ha terminado.

Con Interface Builder

El objeto **self.refreshControl** lo creo con IB

```
- (void)viewDidLoad {
    [super viewDidLoad];

    [self.refreshControl addTarget:self
                            action:@selector(refreshControlValueChanged)
                  forControlEvents:UIControlEventValueChanged];
}

-   (void)refreshControlValueChanged {
    // Descargo los nuevos datos
    . . .

    // Recargar tabla
    [self.tableView reloadData];

    // Terminar el control de refresco
    [self.refreshControl endRefreshing];
}
```