ETSIT UPM

dit UPM

Desarrollo de Apps para iOS

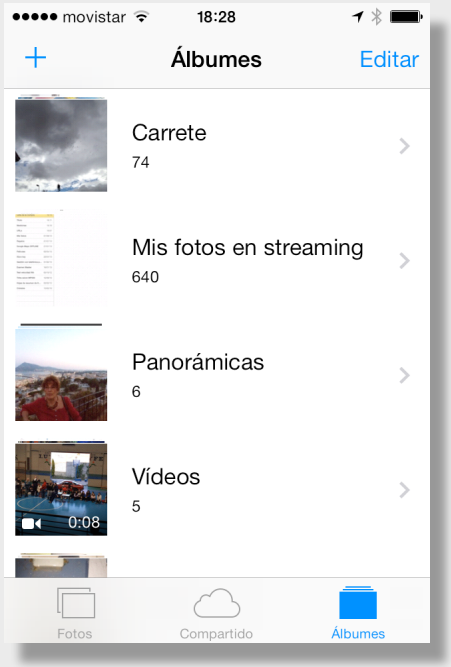
Table Views

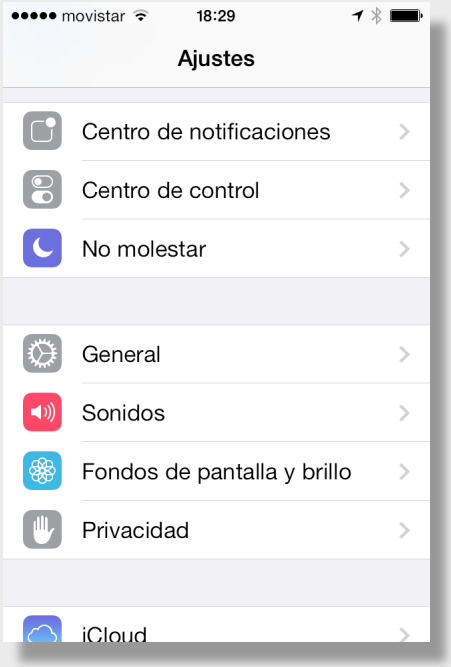
IWEB,LSWC 2014-2015
Santiago Pavón

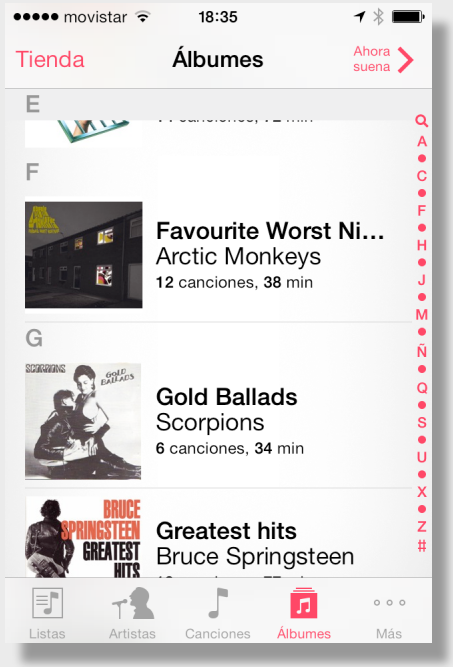
ver: 2014.12.10

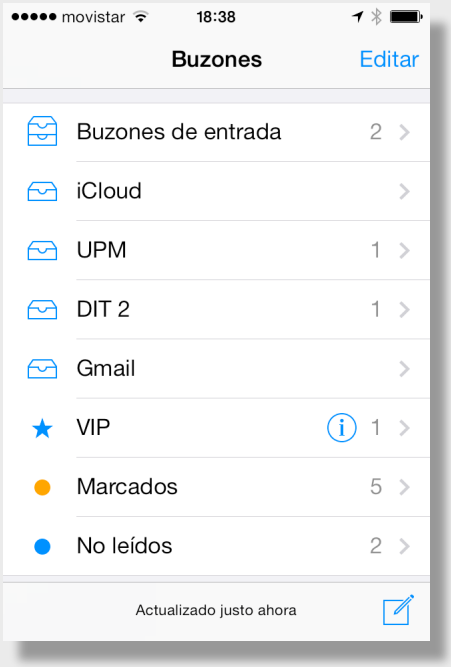
Características de las Tablas

- 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.
- La tabla es un objeto **`UITableView`** y la celda es un objeto **`UITableViewCell`**.
  - **UITableView** deriva de **UIScrollView**.
  - Uso muy eficiente de las celdas reutilizando las celdas no visibles.
- Las tablas pueden ser estáticas (solo en **`UITableViewController`**) o dinámicas.
- Protocolos:
  - Fuente de datos: **`UITableViewDataSource`**.
  - Apariencia y comportamiento: **`UITableViewDelegate`**.









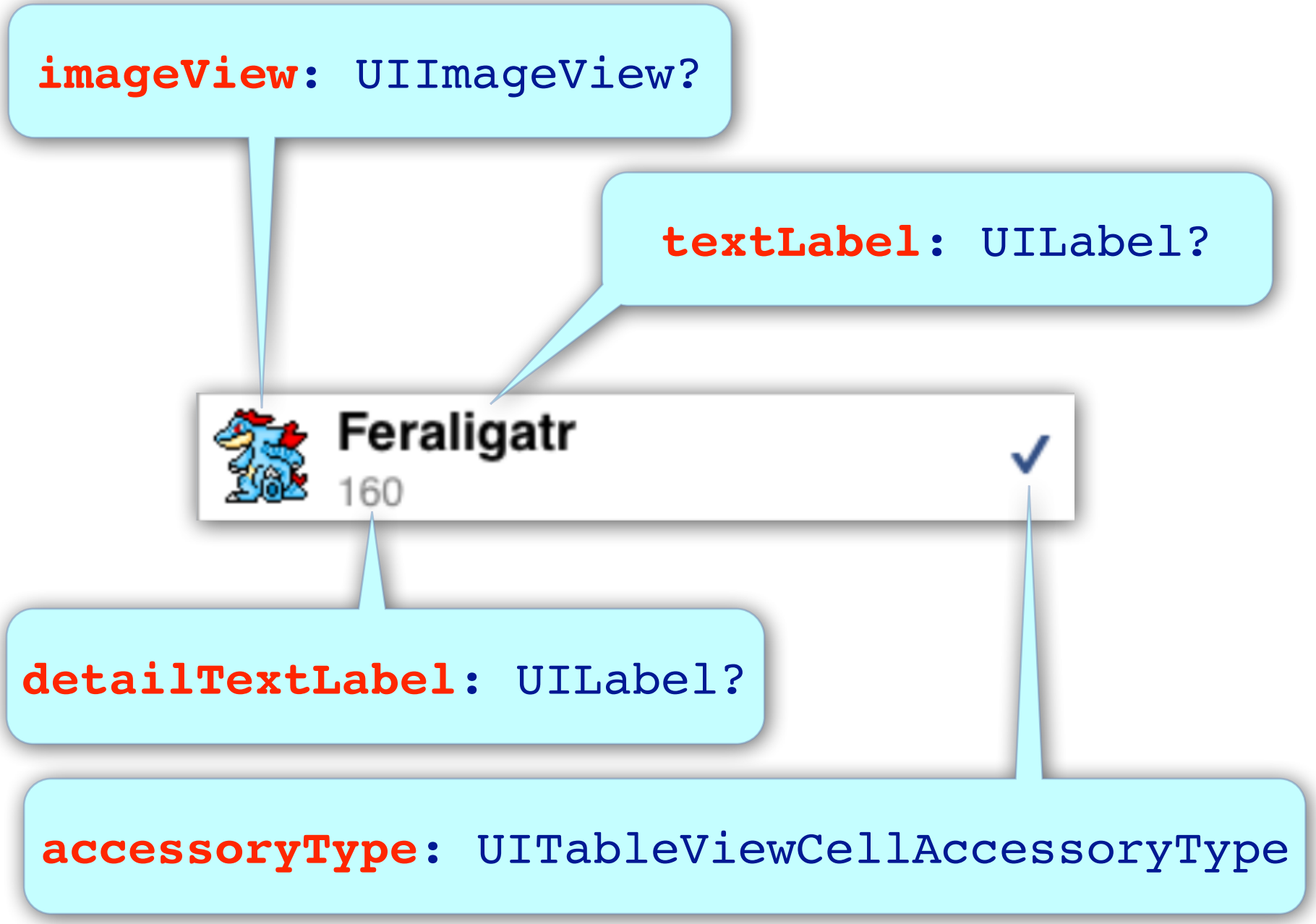
Estilos de Celdas Predefinidos

- **Basic** (UITableViewCellStyle.**Default**)

- **Subtitle** (UITableViewCellStyle.**Subtitle**)

- **Right Detail** (UITableViewCellStyle.**Value1**)

- **Left Detail** (UITableViewCellStyle.**Value2**)



Propiedades de las celdas

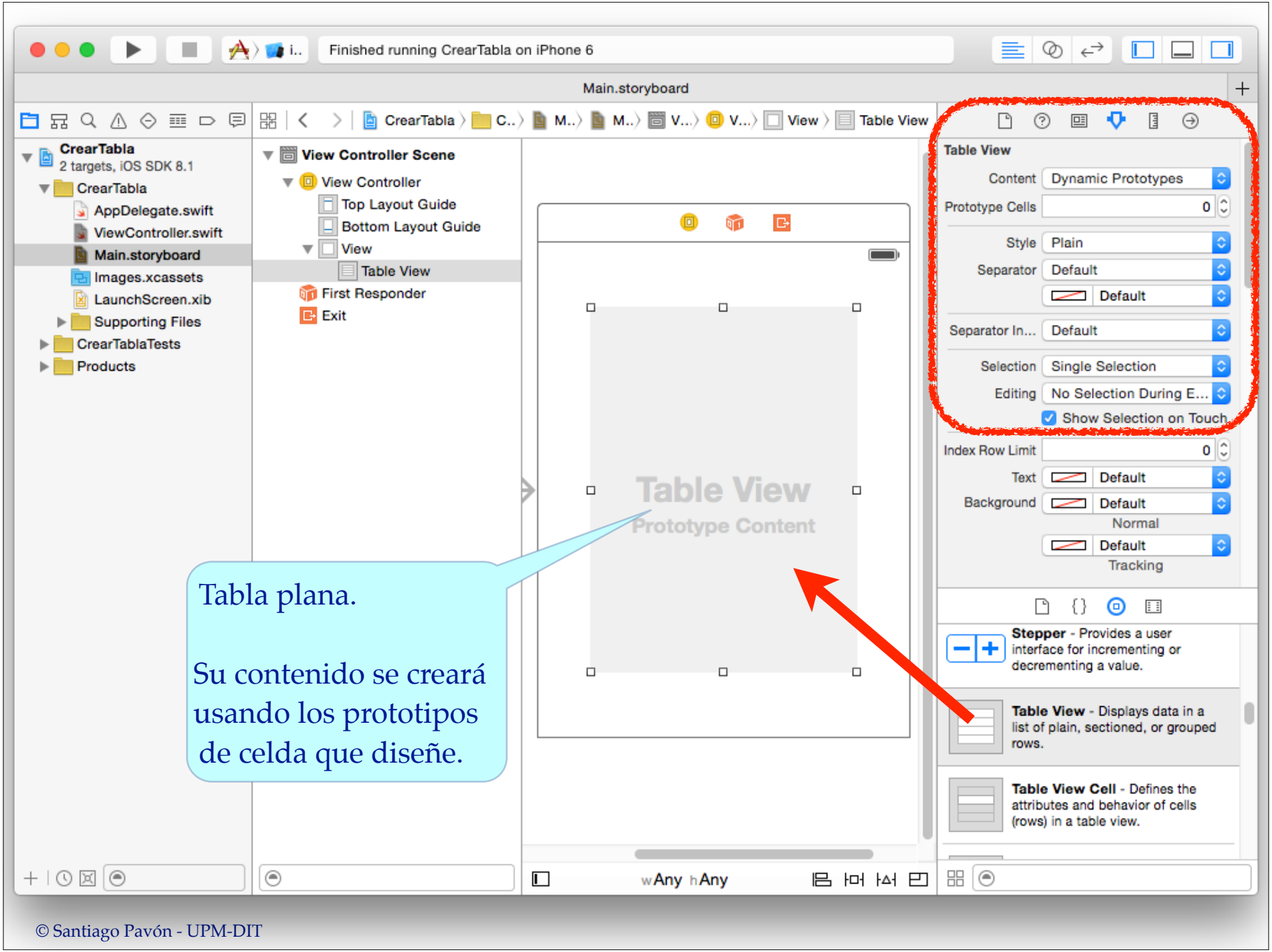
- **`imageView`**
  - Es una **`UIImageView?`** que podemos personalizar cambiando sus propiedades
    - **`image`**, **`highlightedImage`**, . . .
- **`textLabel`**
  - Es una **`UILabel?`** que podemos personalizar cambiando sus propiedades
    - **`text`**, **`font`**, . . .
- **`detailTextLabel`**
  - 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.
- . . .

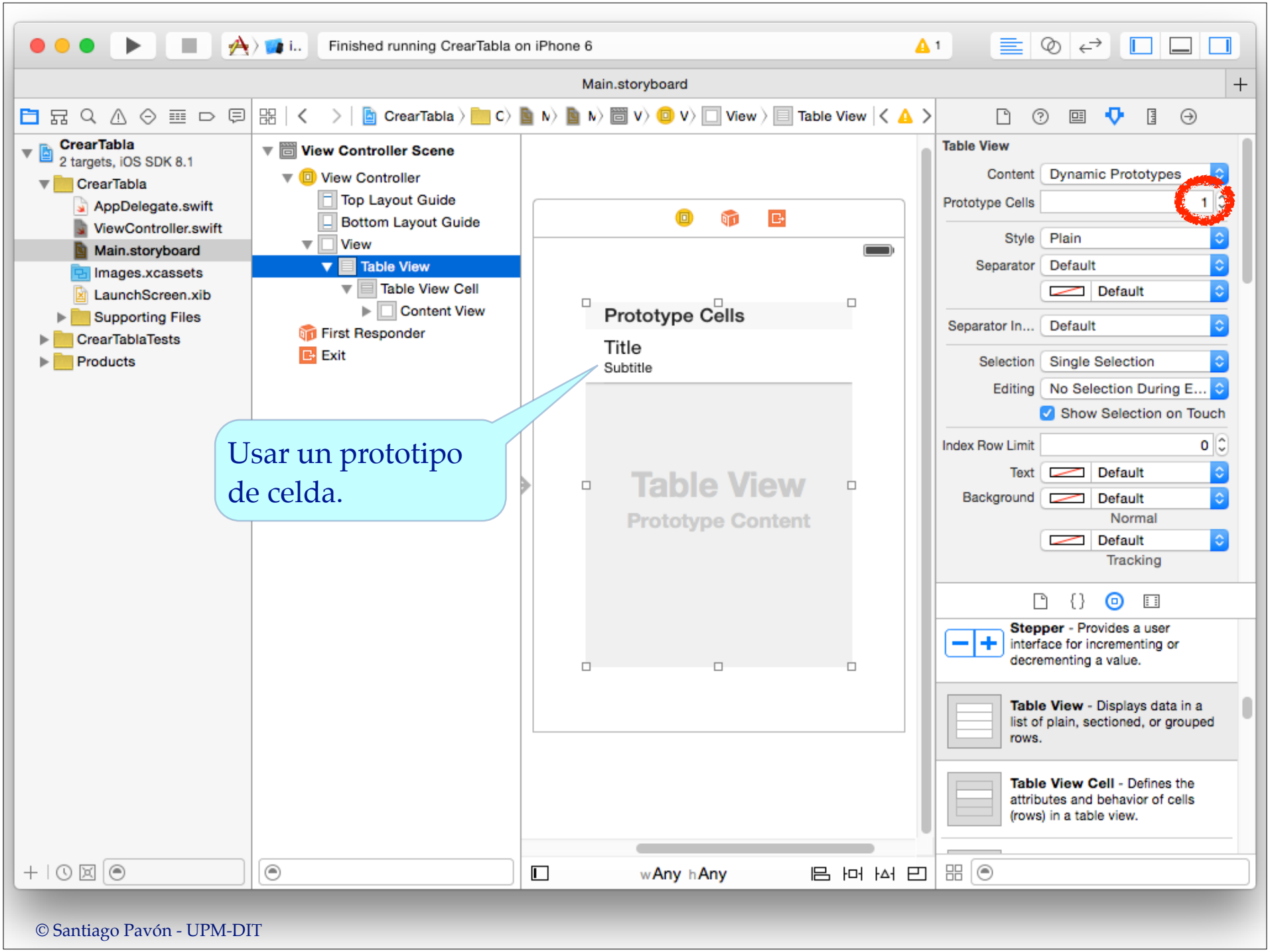
Crear un objeto UITableView

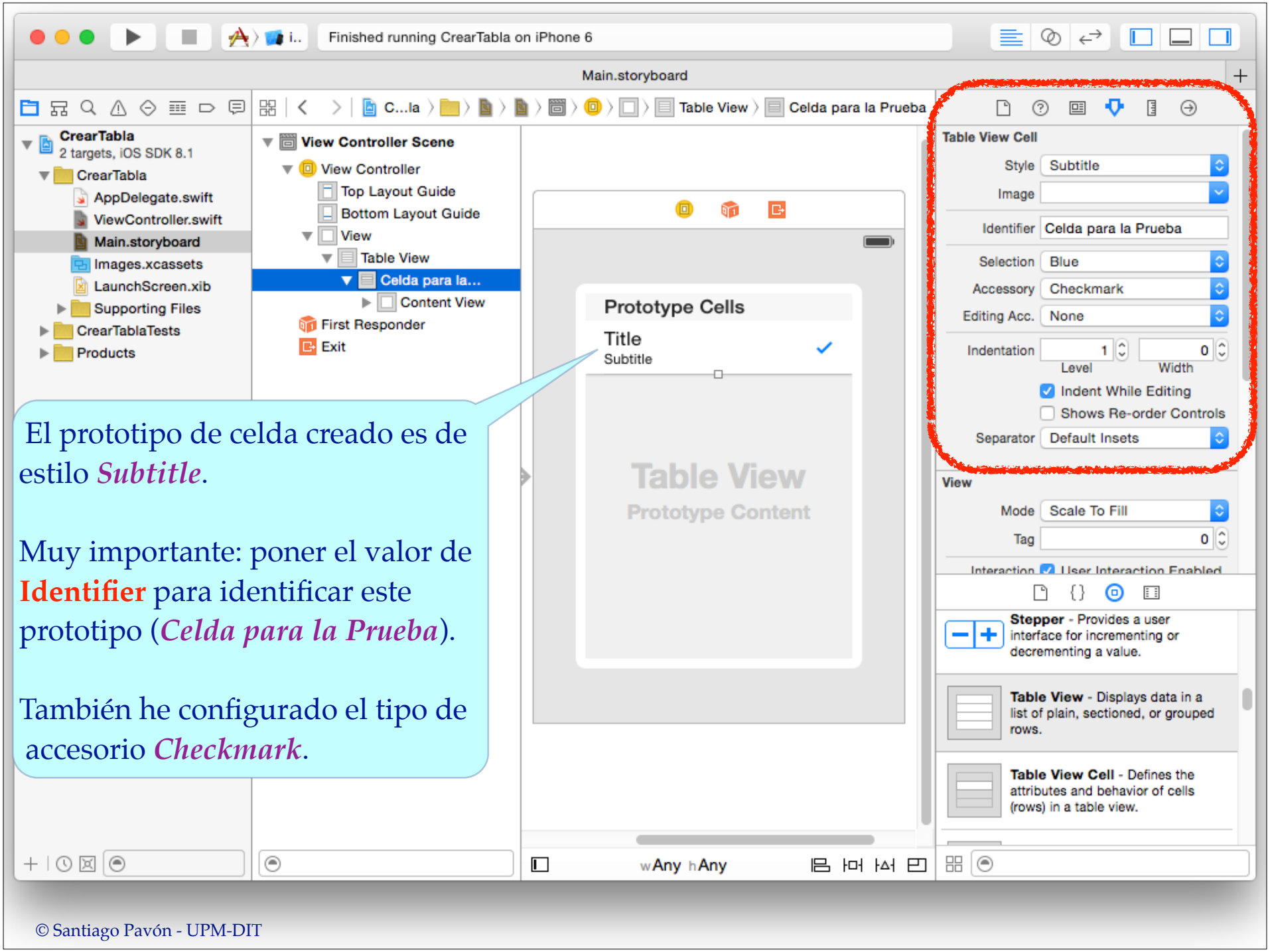
- **Programáticamente:**

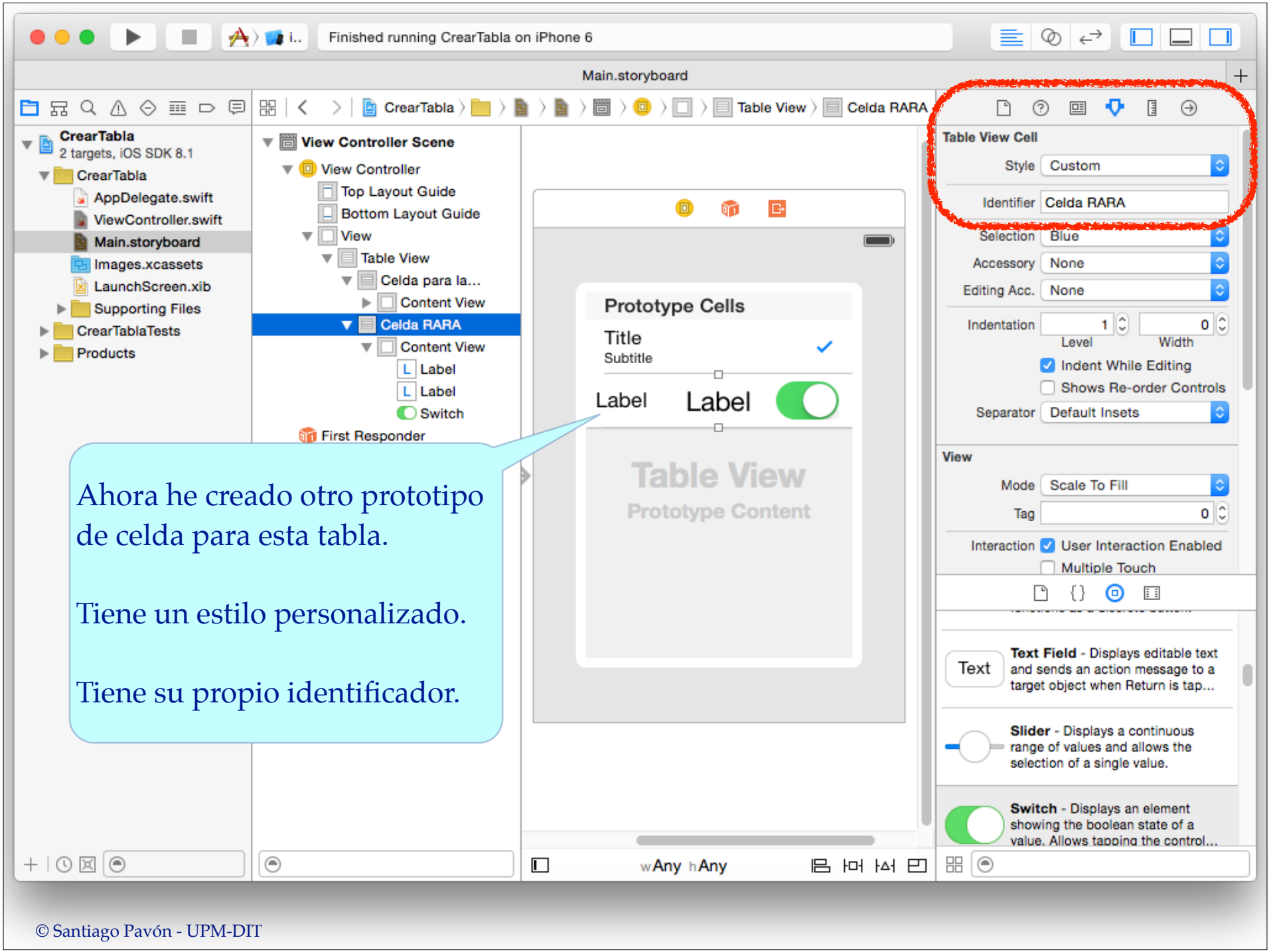
```
let rect = CGRectMake(50, 50, 220, 300)
let tv: UITableView = UITableView(
                                  frame: rect,
                                  style: UITableViewStyle.Plain)
view.addSubview(tv)
```

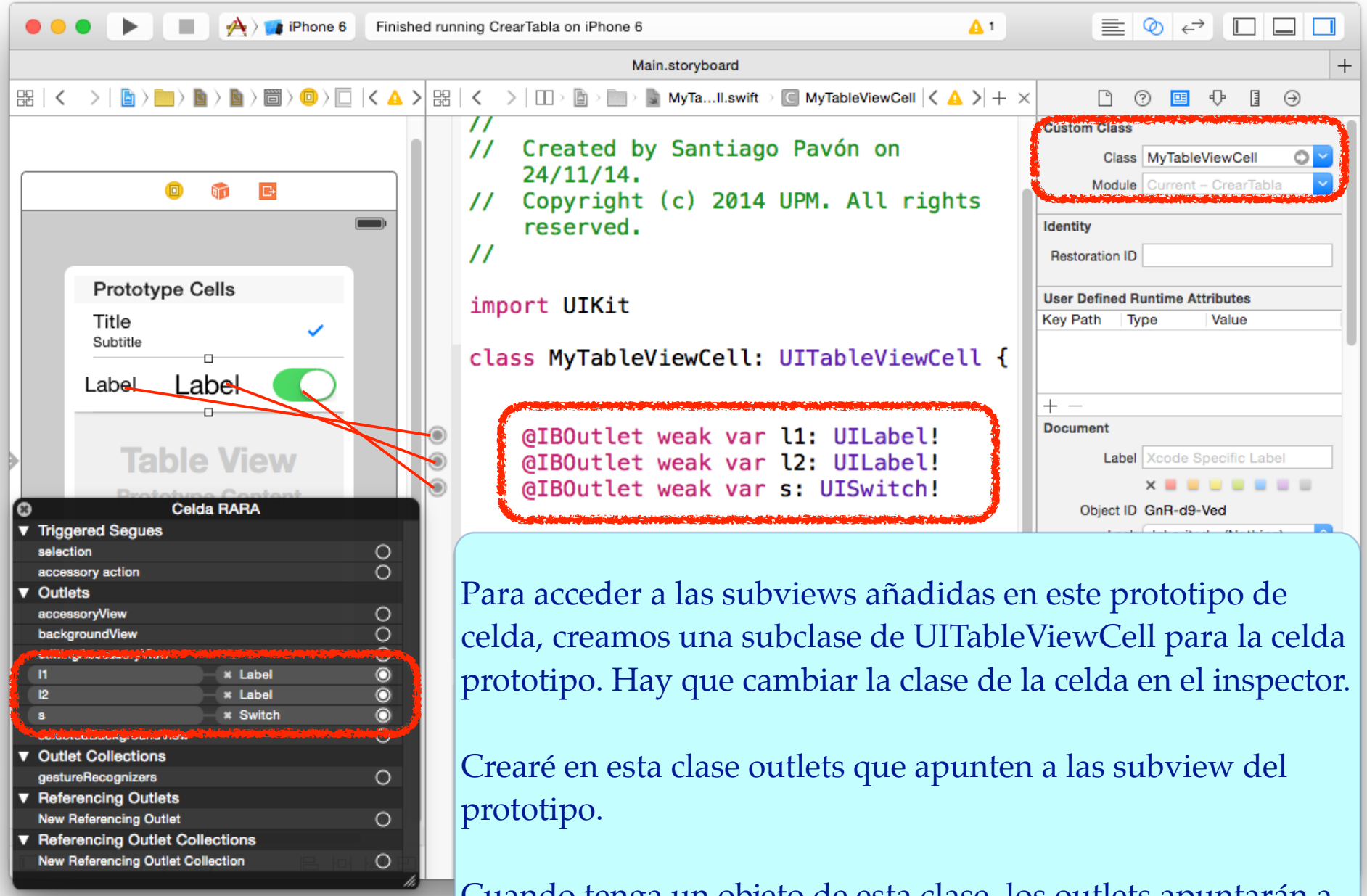
- **Interface Builder y Storyboard:**
  - Arrastrar un objeto Table View desde la librería de objetos.
    - Configurar la tabla en el inspector.
      - Contenido: celdas creadas usando prototipos.
        - Las celdas estáticas solo en UITableViewController.
      - Número de prototipos de celdas que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







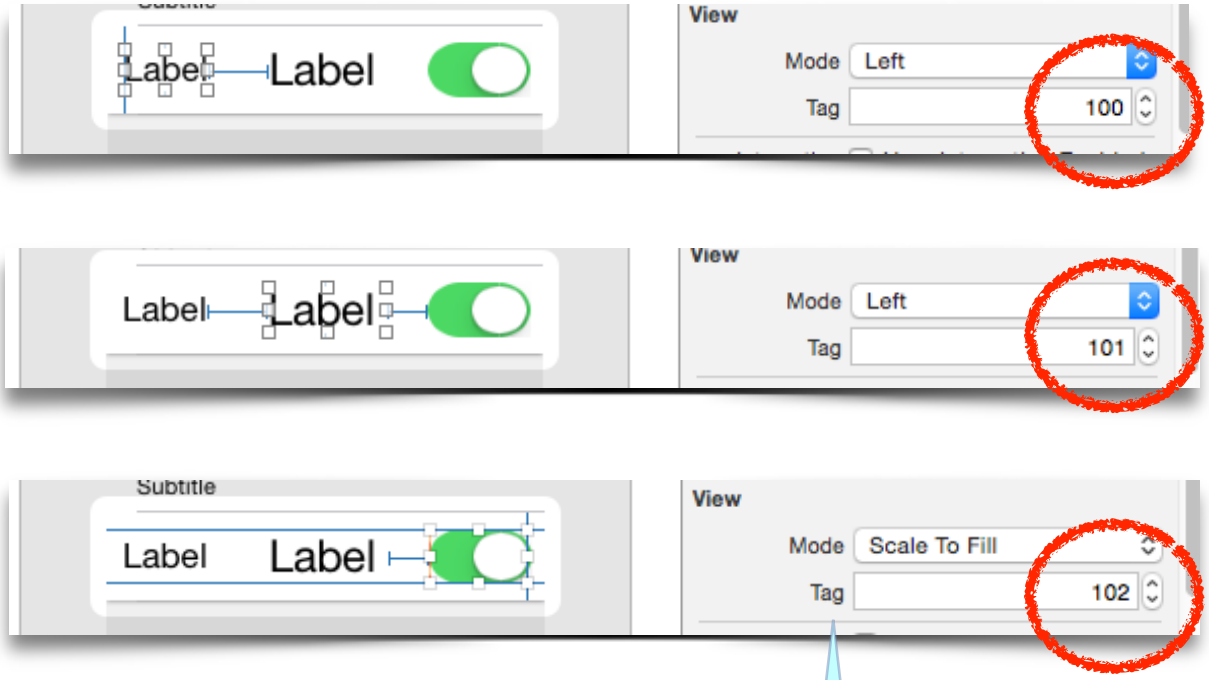




Para acceder a las subviews añadidas en este prototipo de celda, creamos una subclase de UITableViewCell para la celda prototipo. Hay que cambiar la clase de la celda en el inspector.

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

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



Otra forma de acceder a las subviews de este prototipo de celda personalizado, pero sin crear una nueva clase, es asignar un **tag** distinto a cada subview.

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

```
let v: UIView? = cell.viewWithTag(100)
```

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**

```
func tableView(tableView: UITableView,
     numberOfRowsInSection section: Int) -> Int

func tableView(tableView: UITableView,
     cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell

optional func numberOfSectionsInTableView(tableView: UITableView)
                                                  -> Int

optional func tableView(tableView: UITableView,
              titleForHeaderInSection section: Int) -> String?

optional func tableView(tableView: UITableView,
              titleForFooterInSection section: Int) -> String?
```

- **Índice**

```
optional func sectionIndexTitlesForTableView(tableView: UITableView)
                                                -> [AnyObject]!

optional func tableView(tableView: UITableView,
              sectionForSectionIndexTitle title: String,
              atIndex index: Int) -> Int
```

- **Añadir o borrar filas**

```
optional func tableView(tableView: UITableView,
              commitEditingStyle editingStyle: UITableViewCellEditingStyle,
              forRowAtIndexPath indexPath: NSIndexPath)

optional func tableView(tableView: UITableView,
              canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool
```

- **Reordenar las filas**

```
optional func tableView(tableView: UITableView,
              canMoveRowAtIndexPath indexPath: NSIndexPath) -> Bool

optional func tableView(tableView: UITableView,
              moveRowAtIndexPath sourceIndexPath: NSIndexPath,
              toIndexPath destinationIndexPath: NSIndexPath)
```

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 (**`Int`**).

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 saca de la cola de reutilización con:

```
func dequeueReusableCellWithIdentifier(_ identifier: String,
          forIndexPath indexPath: NSIndexPath) -> AnyObject
```

- Si la cola está vacía, construye una nueva celda copiando el prototipo.

```
func dequeueReusableCellWithIdentifier(_ identifier: String)
                                              -> AnyObject?
```

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

```
init(style style: UITableViewCellStyle,
     reuseIdentifier reuseIdentifier: String?)
```

Identificar tipo de celda o prototipo

```
override func tableView(tableView: UITableView,
                        cellForRowAtIndexPath indexPath: NSIndexPath)
                        -> UITableViewCell {

  // Saca una celda de la cola para reutilizar
  let cell = tableView.dequeueReusableCellWithIdentifier("My Cell Id",
                                                         forIndexPath: indexPath)
                              as MyTableViewCell

  // Miro el indexPath para saber que debo mostrar en esta celda
  let row = indexPath.row
  let section = indexPath.section

  // Configurar la celda
  cell.imageView?.image = almacen.getPhoto(row)
  cell.imageView?.highlightedImage = almacen.getHPhoto(row)
  cell.textLabel?.text = almacen.getName(row)
  cell.accessoryType = UITableViewCellAccessoryType.Checkmark

  // Devolver la celda
  return cell
}
```

UITableView - Actualizaciones

```
func reloadData()

func insertSections(_ sections: NSIndexSet,
   withRowAnimation animation: UITableViewRowAnimation)

func deleteSections(_ sections: NSIndexSet,
   withRowAnimation animation: UITableViewRowAnimation)

func reloadSections(_ sections: NSIndexSet,
   withRowAnimation animation: UITableViewRowAnimation)

func insertRowsAtIndexPaths(_ indexPaths: [AnyObject],
          withRowAnimation animation: UITableViewRowAnimation)

func deleteRowsAtIndexPaths(_ indexPaths: [AnyObject],
          withRowAnimation animation: UITableViewRowAnimation)


func reloadRowsAtIndexPaths(_ indexPaths: [AnyObject],
          withRowAnimation animation: UITableViewRowAnimation)
. . .
```

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, <u>despúes de seleccionar</u> una fila se invoca:

```
func tableView(_ tableView: UITableView,
     didSelectRowAtIndexPath indexPath: NSIndexPath)
```

    - 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.

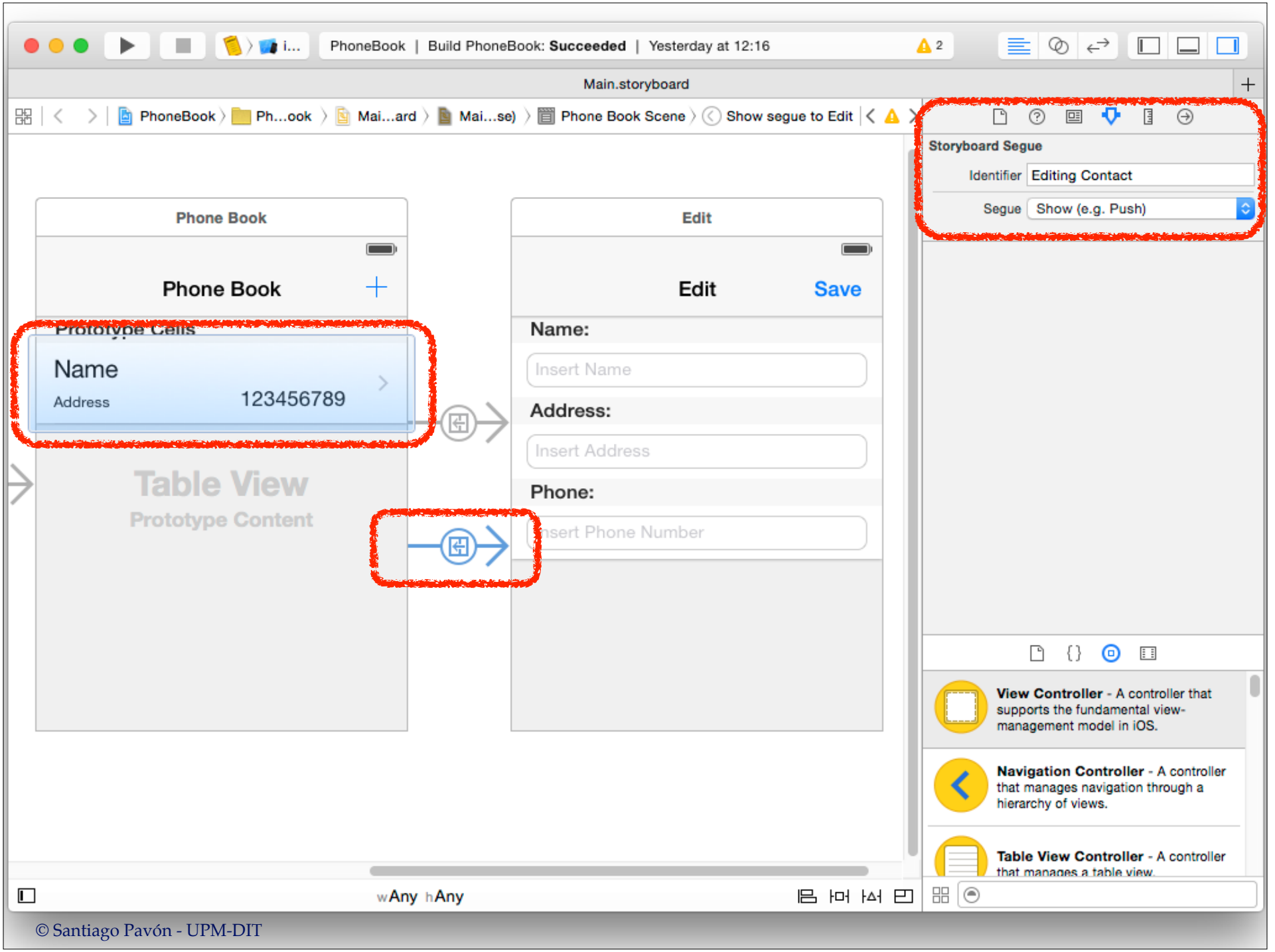
- También podemos crear un segue que se dispare cuando se selecciona una fila de la tabla.
  - Asignaremos un identificador al segue.
  - Adaptaremos el método **`prepareForSegue:sender:`** para:
    - Consultar que celda está seleccionada para saber dónde hemos pulsado.
    - Configurar el VC destino del segue.

Ejemplo

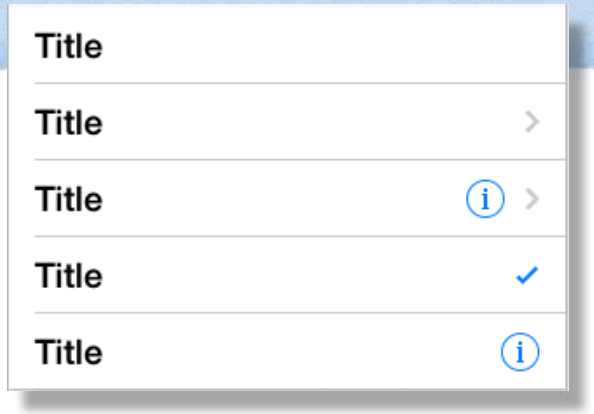
```
override func tableView(tableView: UITableView,
                        didSelectRowAtIndexPath indexPath: NSIndexPath) {

  // Creo un objeto VC nuevo cargandolo del storyboard
  if let ivc = storyboard?.instantiateViewControllerWithIdentifier("ID")
         as? InfoViewController {

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

    // Paso el VC al Navigation Controller para que lo muestre
    navigationController?.pushViewController(ivc, animated: true)
  }
}
```

Ejemplo

```
override func prepareForSegue(segue: UIStoryboardSegue, sender: AnyObject?) {
    if segue.identifier == "Editing Contact" {
        let ecvc = segue.destinationViewController as EditContactViewController
        let cell = sender as UITableViewCell
        if let row = tableView.indexPathForCell(cell)?.row {
           ecvc.contact = contactsTable[row]
        }
    } else if segue.identifier == "Adding Contact" {
        let ecvc = segue.destinationViewController as EditContactViewController
        var newContact = Contact(name: "", address: "", phone: "")
        contactsTable.insertContact(newContact, atindex:0)
        let ip = NSIndexPath(forRow: 0, inSection: 0)
        tableView.insertRowsAtIndexPaths([ip], withRowAnimation:.Automatic)
        ecvc.contact = newContact
    }
}
```



Accesorios



- Controles accesorios usados por las celdas

```
UITableViewCellAccessoryType.None
UITableViewCellAccessoryType.Checkmark
UITableViewCellAccessoryType.DetailButton
UITableViewCellAccessoryType.DisclosureIndicator
UITableViewCellAccessoryType.DetailDisclosureButton
```

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

```
func tableView(_ tableView: UITableView,
    accessoryButtonTappedForRowWithIndexPath indexPath: NSIndexPath)
```

- 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.

Tamaño de las Celdas antes de iOS 8

- 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.
- Programáticamente, podemos indicar cual es la altura de las celdas asignando un valor a **`tableView.rowHeight`**.
- 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.

Tamaño de las Celdas con iOS 8

- Con iOS 8 se simplifica el calculo del tamaño de las celdas.
  - Especialmente cuando las celdas pueden ser de distintos tamaños.
  - Solo hay que asegurarse de que las restricciones de autolayout añadidas a la celda (en su prototipo) determinán cuál es su altura.
    - El ancho lo determina la tableView.
  - Ya no hay que usar los métodos que tiene UITableView para calcular el tamaño real de cada celda, o estimarlo, en función de su IndexPath.
    - Las restricciones añadidas junto con el tamaño intrínseco de las subviews de cada celda determinan su tamaño.
  - Pero hay que añadir programáticamente las siguientes sentencias:
    - Decir a la TableView que las celdas calculan su propio tamaño:

      ```
      tableview.rowHeight = UITableViewAutomaticDimension
      ```

    - Y proporcionar un tamaño estimado de celda distinto de cero.
      - Este valor debe ser cercano al real para evitar saltos al hacer scroll.

      ```
      tableview.estimatedRowHeight = 100
      ```

Editar una Tabla

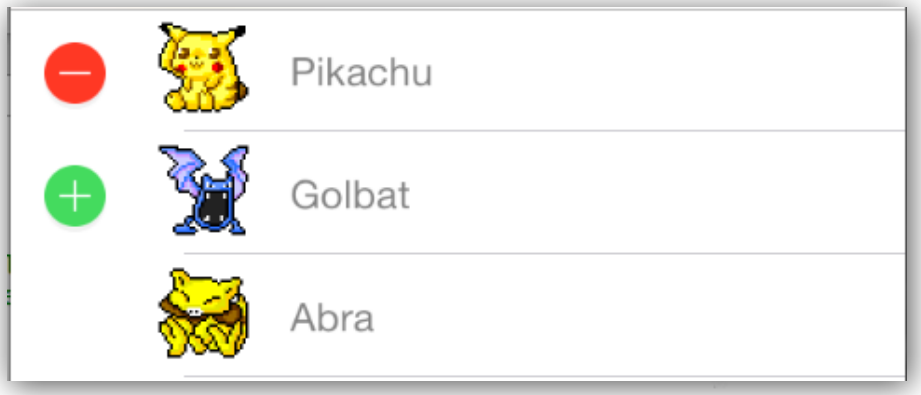
- Preguntar al **data source** si la celda es editable.

```
func tableView(tableView: UITableView,
     canEditRowAtIndexPath indexPath: NSIndexPath) -> Bool
```

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

```
func tableView(_ tableView: UITableView,
     editingStyleForRowAtIndexPath indexPath: NSIndexPath)
     -> UITableViewCellEditingStyle
```

  - Se muestra un icono indicando el tipo de edición



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

```
func tableView(tableView: UITableView,
     commitEditingStyle editingStyle: UITableViewCellEditingStyle,
     forRowAtIndexPath indexPath: 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:

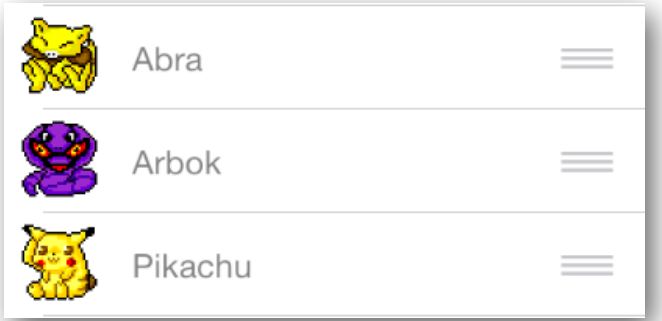
```
func tableView(tableView: UITableView,
     canMoveRowAtIndexPath indexPath: NSIndexPath)
     -> Bool
```

    - Para actualizar el data source moviendo una celda a otra posición:

```
func tableView(tableView: UITableView,
     moveRowAtIndexPath sourceIndexPath: NSIndexPath,
     toIndexPath destinationIndexPath: NSIndexPath)
```

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

```
var showsReorderControl: Bool
```



UITableViewController

- Es una clase derivada de **`UIViewController`**
  - que tiene una **`UITableView`** como su view.
    - La propiedad **`view`** y **`tableView`** apuntan al mismo objeto: la 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.

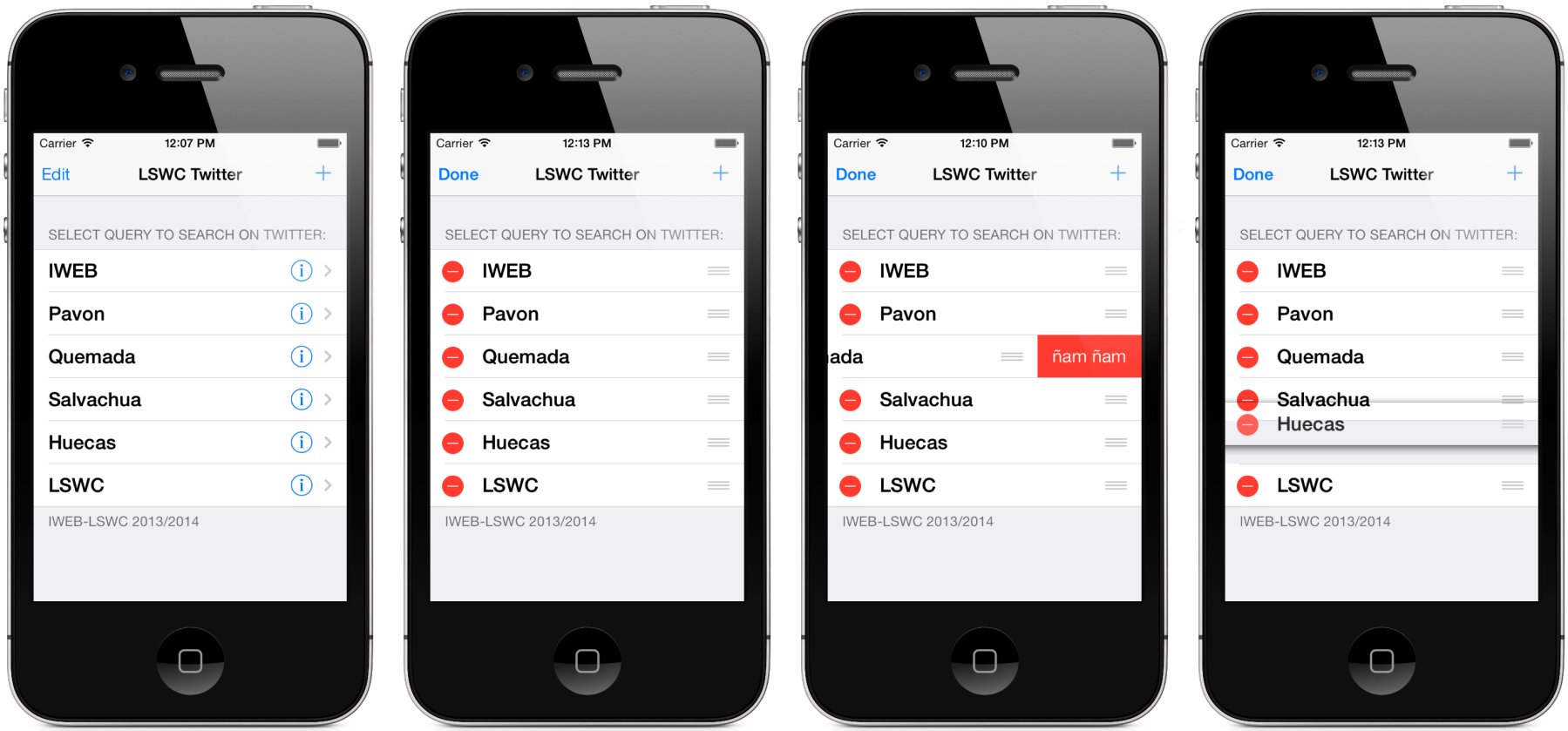
`UITableViewController` - Modo Edición

- 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:

```
func setEditing(_ editing: Bool,
          animated animated: Bool)
```

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

Ejemplo: editar una tabla



```
override func viewDidLoad() {
    super.viewDidLoad()

    // display the Edit button in the navigation bar
    self.navigationItem.leftBarButtonItem = self.editButtonItem()

    loadQueries()
}


override func tableView(tableView: UITableView,
                titleForDeleteConfirmationButtonForRowAtIndexPath
                        indexPath: NSIndexPath) -> String! {
    return "ñam ñam"
}
```

```
override func tableView(tableView: UITableView,
                        commitEditingStyle editingStyle: UITableViewCellEditingStyle,
                        forRowAtIndexPath indexPath: NSIndexPath) {

    if editingStyle == .Delete {

        // Borro el dato de esa fila de mi modelo.
        queries.removeAtIndex(indexPath.row)

        // Actualizar lo que muestra la table view
        tableView.deleteRowsAtIndexPaths([indexPath],
                                withRowAnimation: .Fade)

        saveQueries() // persistencia

    } else if editingStyle == .Insert) {

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

}
```

```
override func tableView(tableView: UITableView,
                        moveRowAtIndexPath fromIndexPath: NSIndexPath,
                        toIndexPath: NSIndexPath) {

    let obj = queries[fromIndexPath.row]

    queries.removeAtIndex(fromIndexPath.row)
    queries.insert(obj, atIndex: toIndexPath.row)

    saveQueries()
}


override func tableView(tableView: UITableView,
                        canMoveRowAtIndexPath indexPath: NSIndexPath) -> Bool {
    return true
}
```

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.

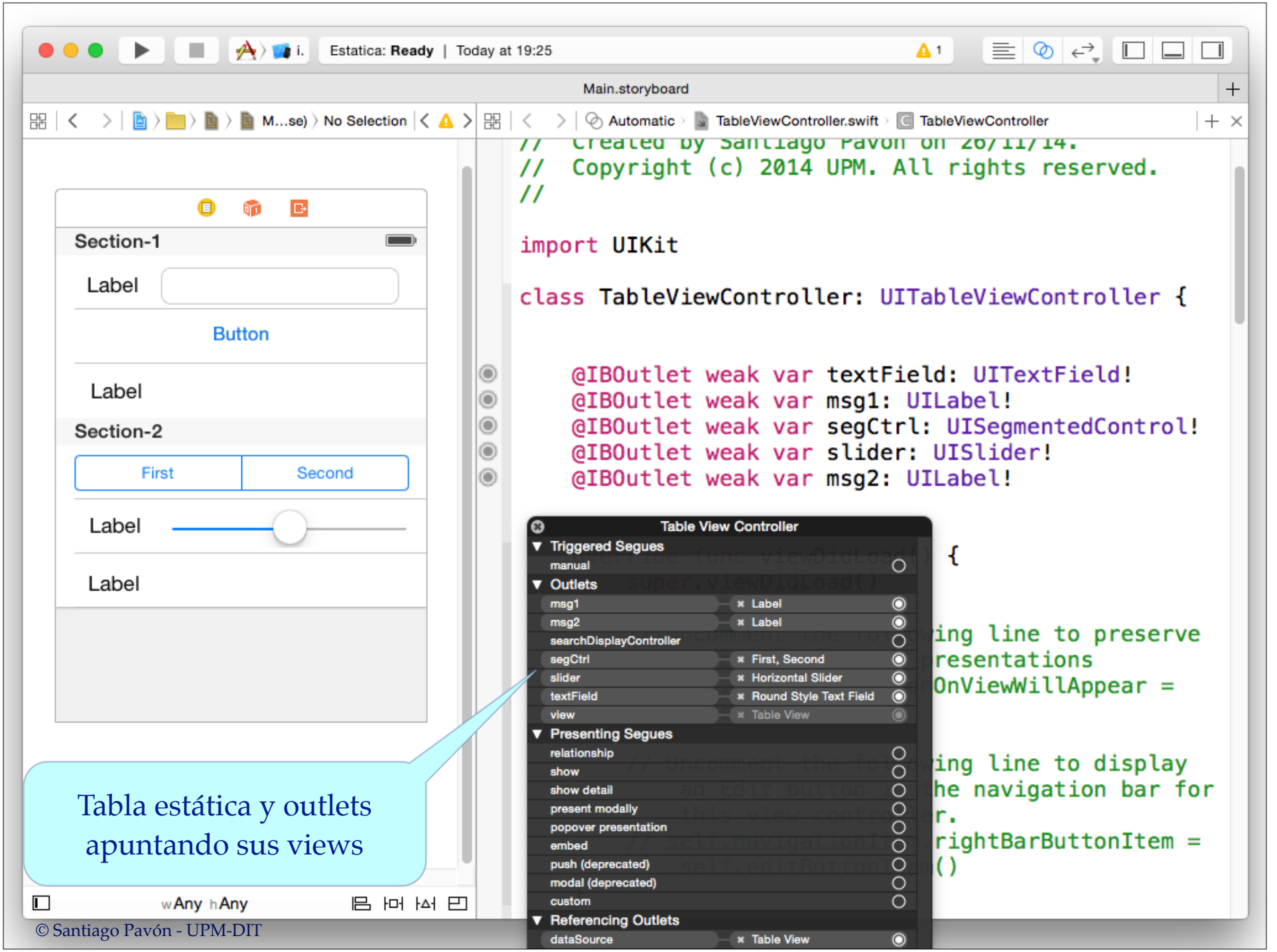
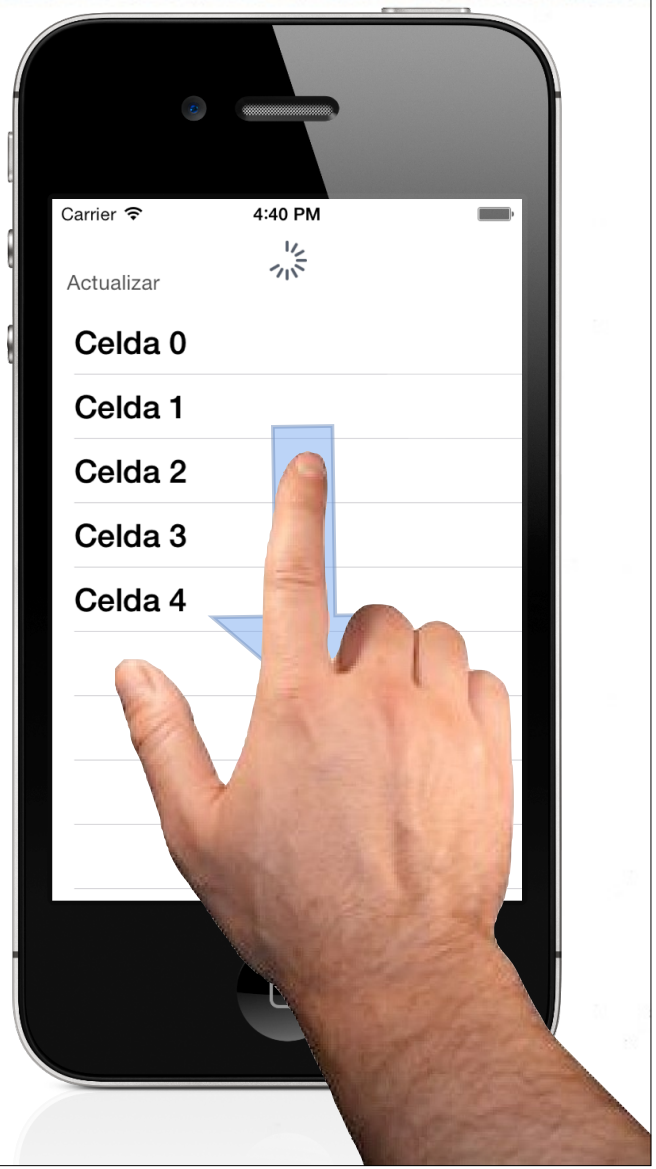


Tabla estática y outlets apuntando sus views

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

```
override func viewDidLoad() {
    super.viewDidLoad()

    refreshControl = UIRefreshControl()

    refreshControl?.addTarget(self,
                              action:"refreshMyTable",
                    forControlEvents:.ValueChanged)
}


func refreshMyTable() {
   // Descargo los nuevos datos
   . . .

   // Recargar tabla
   tableView.reloadData()

   // Terminar el control de refresco
   refreshControl?.endRefreshing()
}
```

Creo un objeto **UIRefreshControl** y lo asigno a self.refreshControl.

Target - Action

.ValueChanged es el evento que se genera al estirar.

Indicar que el refresco ha terminado.

Con Interface Builder

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

```
override func viewDidLoad() {
    super.viewDidLoad()

    refreshControl?.addTarget(self,
                          action:"refreshMyTable",
                forControlEvents:.ValueChanged)
}


func refreshMyTable() {
   // Descargo los nuevos datos
   . . .

   // Recargar tabla
   tableView.reloadData()

   // Terminar el control de refresco
   refreshControl?.endRefreshing()
}
```