# Desarrollo de Apps para iOS
# Table Views

IWEB 2017-2018
Santiago Pavón

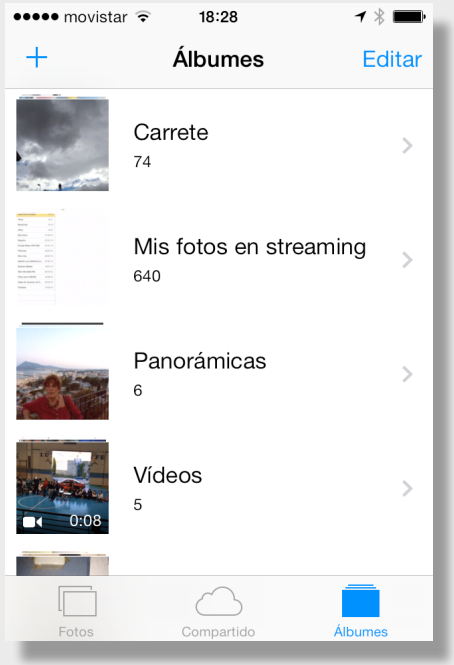
ver: 2017.09.04

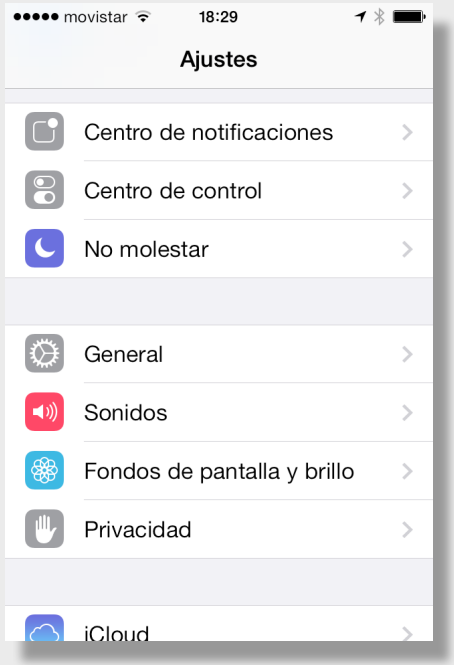
© Santiago Pavón - UPM-DIT

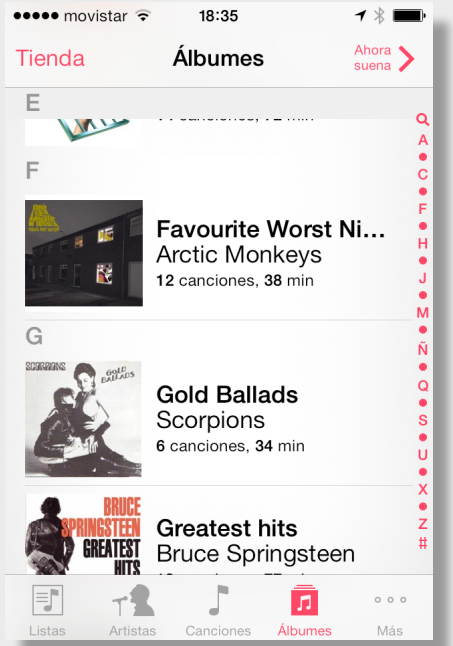
# Características de las Tablas

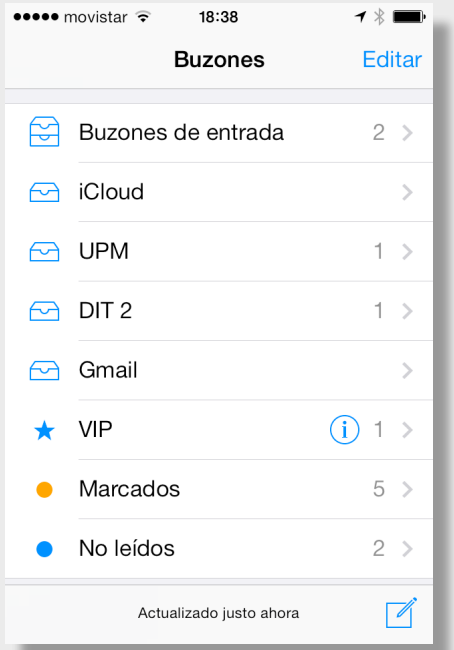
- Mostrar la información en una tabla de una columna.
  - Cada fila es una celda.
- Apariencia:
  - Pueden tener una cabecera y pie de tabla.
  - Se pueden agrupar las celdas en secciones.
    - Cada sección: cabecera, celdas y pie.
  - Existen varios estilos predefinidos de celdas, o podemos crear celdas 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`**.

© Santiago Pavón - UPM-DIT








© Santiago Pavón - UPM-DIT

# Estilos de Celdas Predefinidos

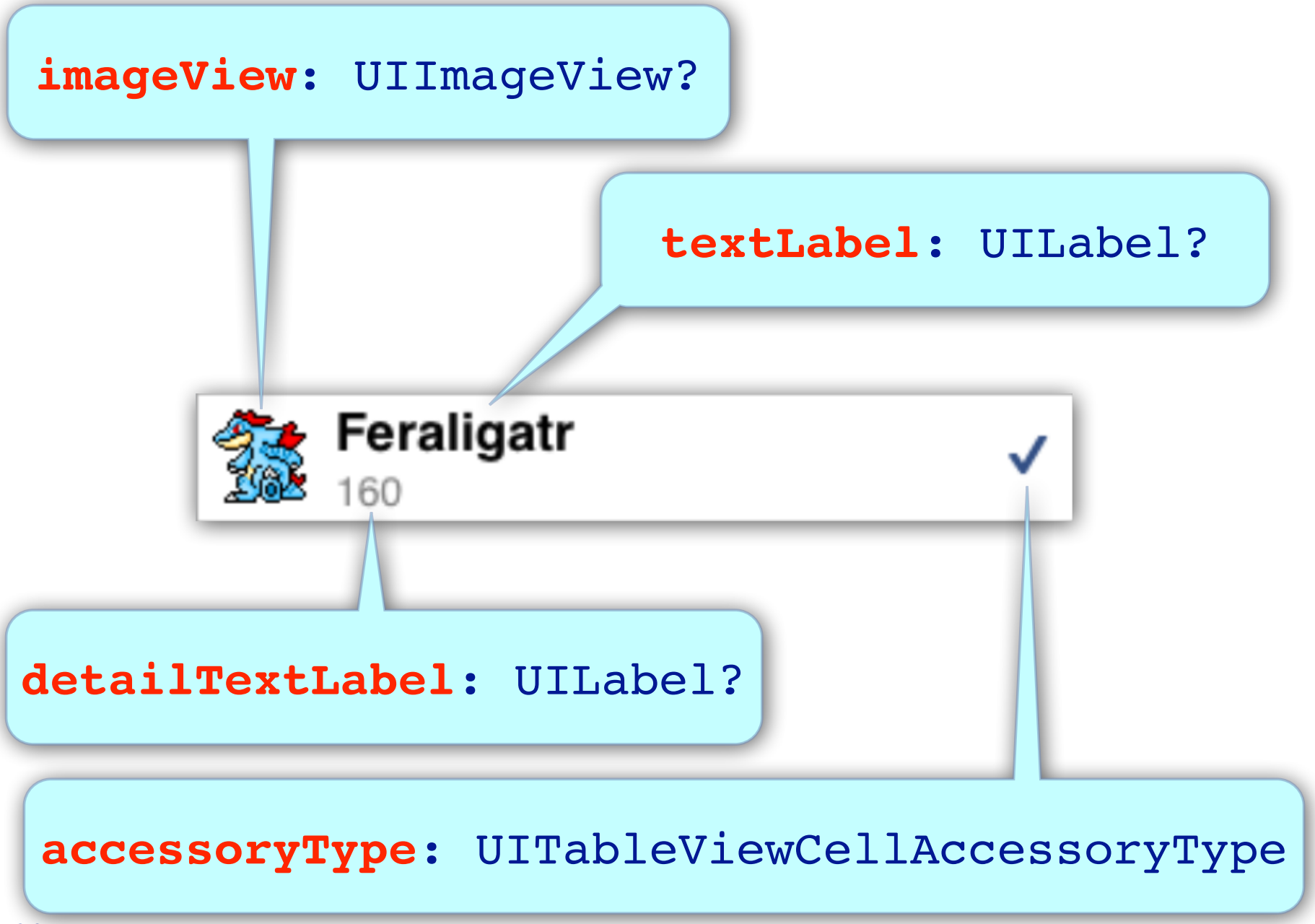
- **Basic** (UITableViewCellStyle.**default**)

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

- **Right Detail** (UITableViewCellStyle.**value1**)

- **Left Detail** (UITableViewCellStyle.**value2**)

© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT

# 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.
- . . .

© Santiago Pavón - UPM-DIT

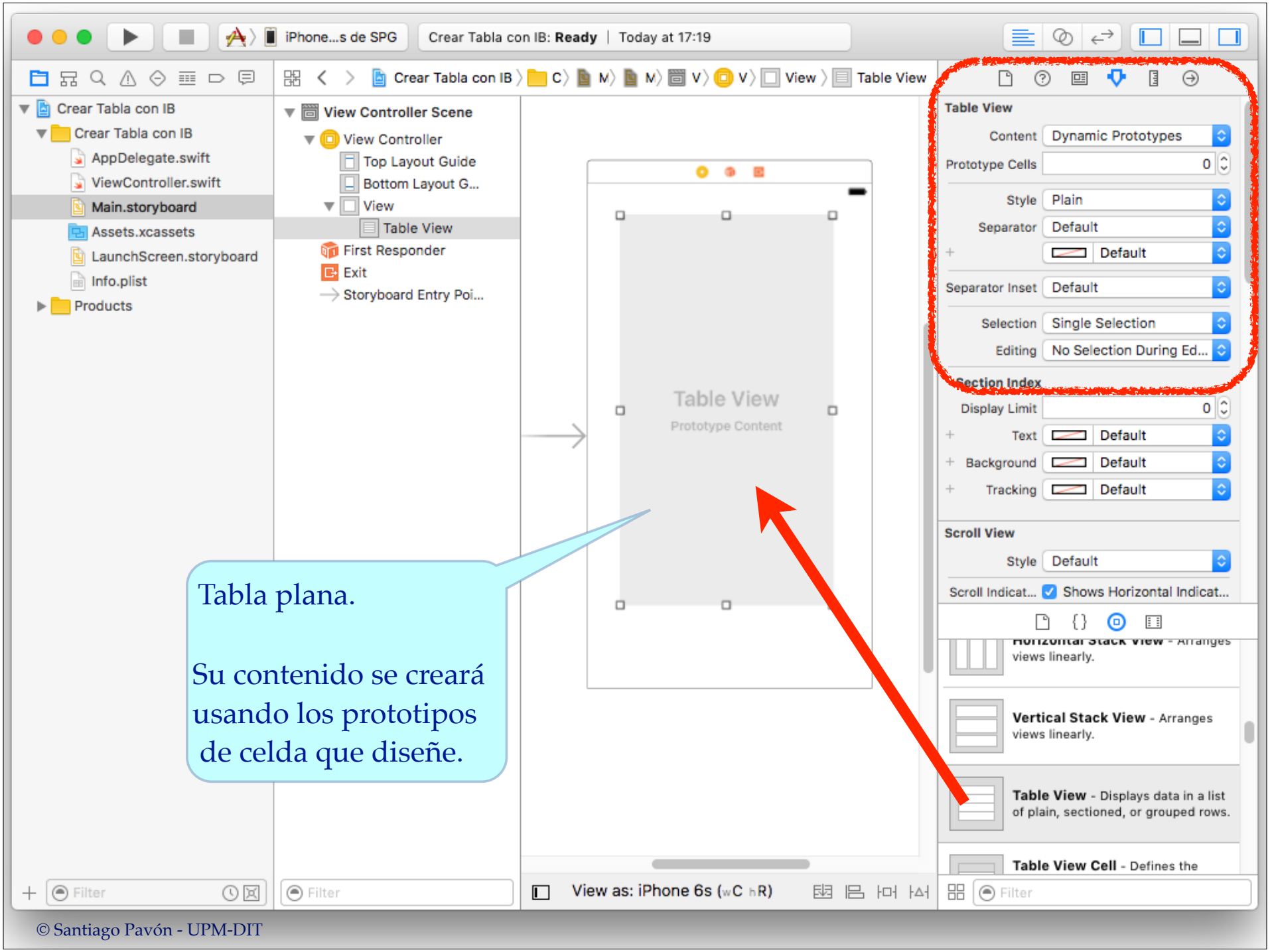
# Crear un objeto UITableView

- **Programáticamente:**

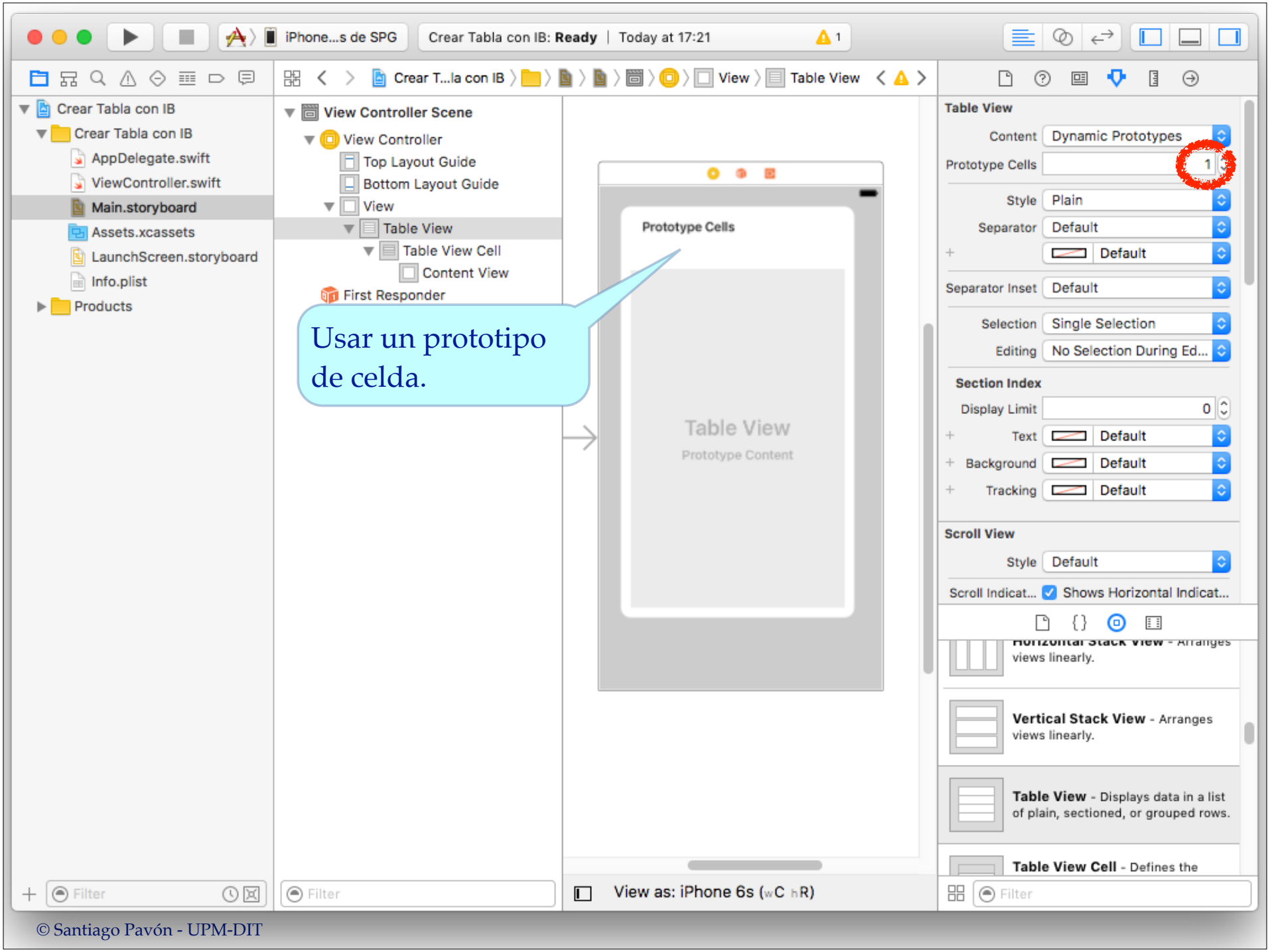
```
let rect = CGRect(x: 50, y: 50, width: 220, height: 300)
let tv = UITableView(frame: rect, style: .plain)
view.addSubview(tv)
```

- **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 se soportan en UITableViewController.
      - Número de prototipos de celdas que queremos usar.
      - Estilo de la tabla: plana, agrupada.
      - ...
    - Asignar su objeto delegado y data source.
    - ...

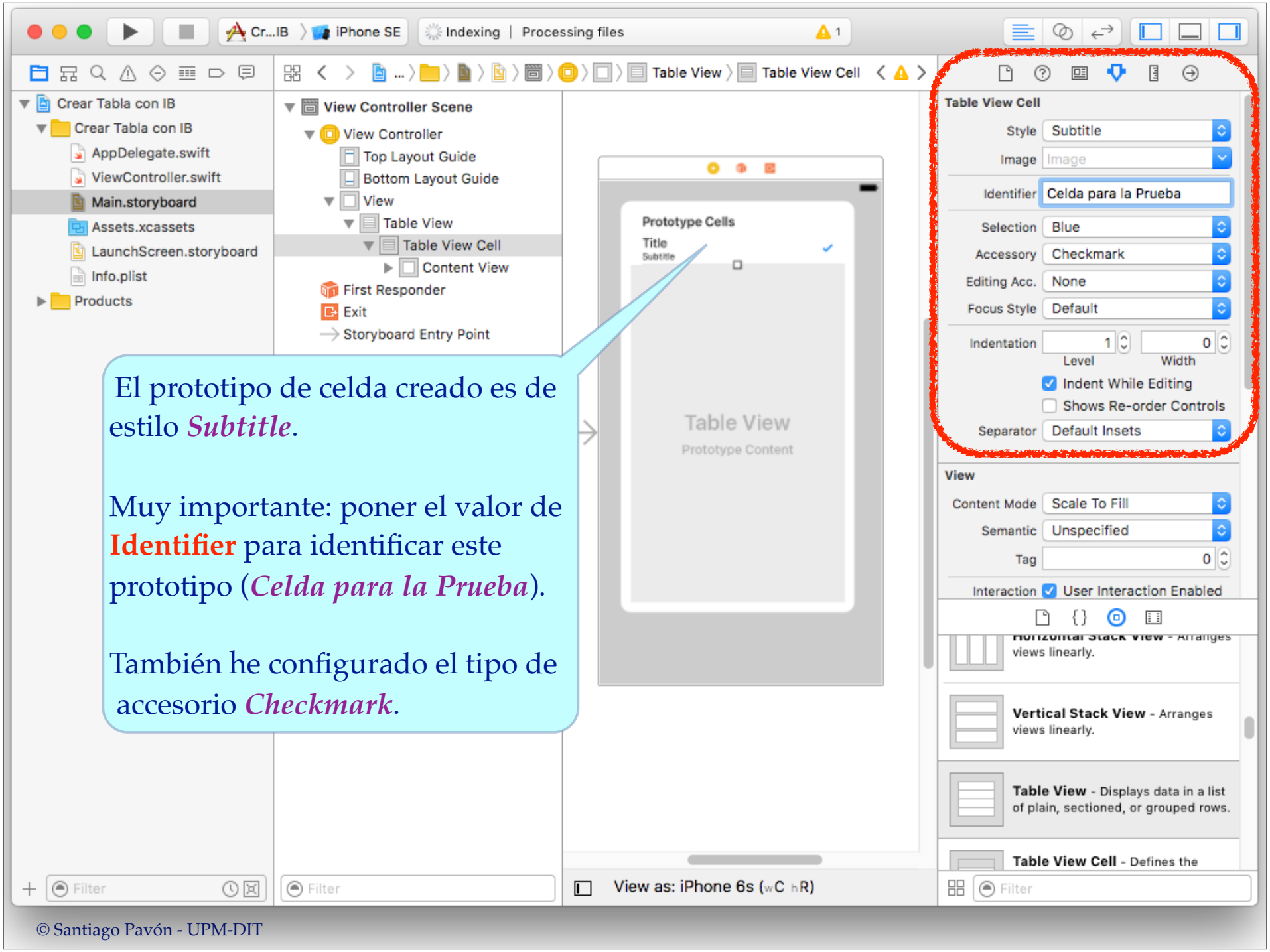
© Santiago Pavón - UPM-DIT



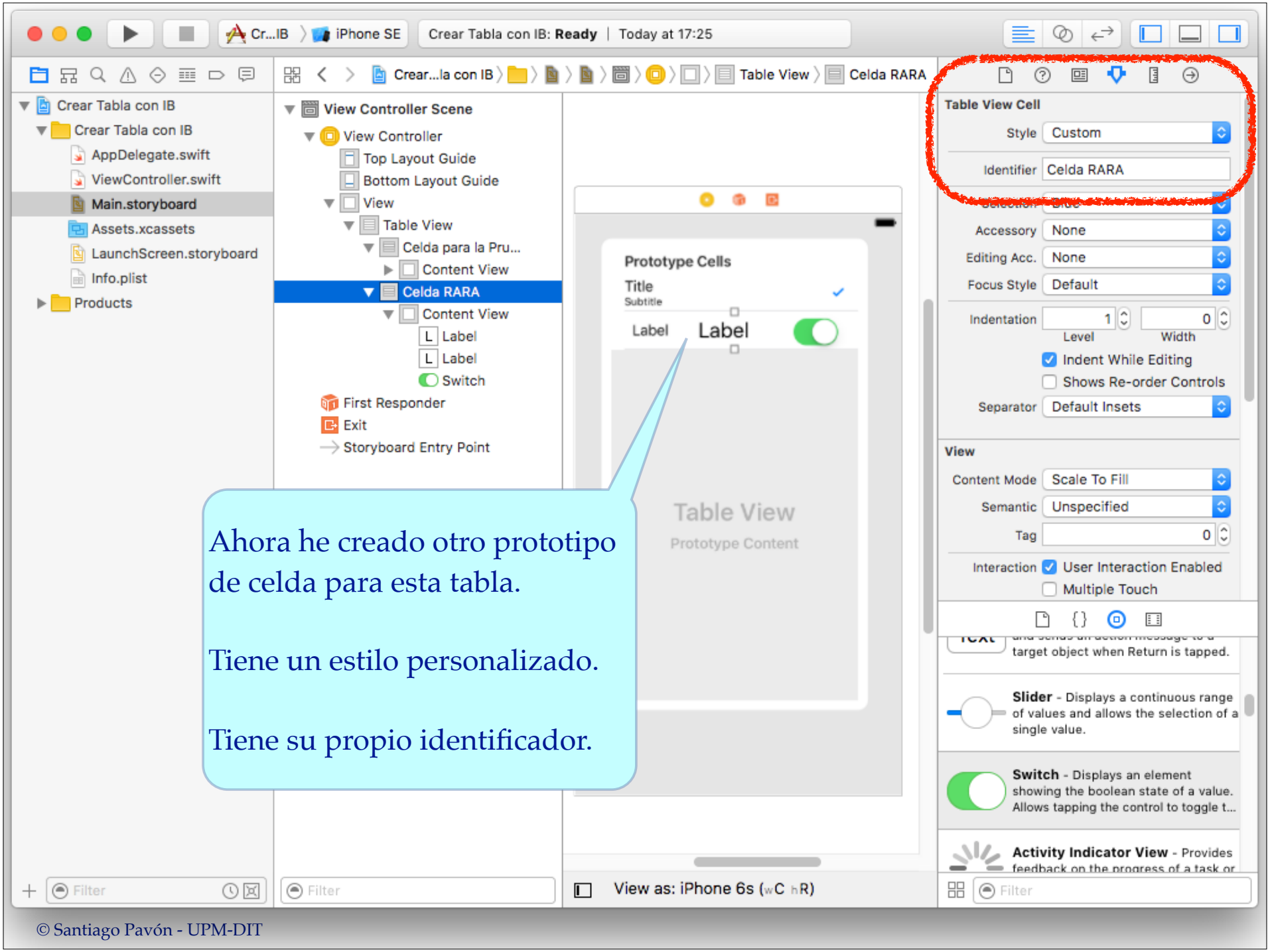

© Santiago Pavón - UPM-DIT



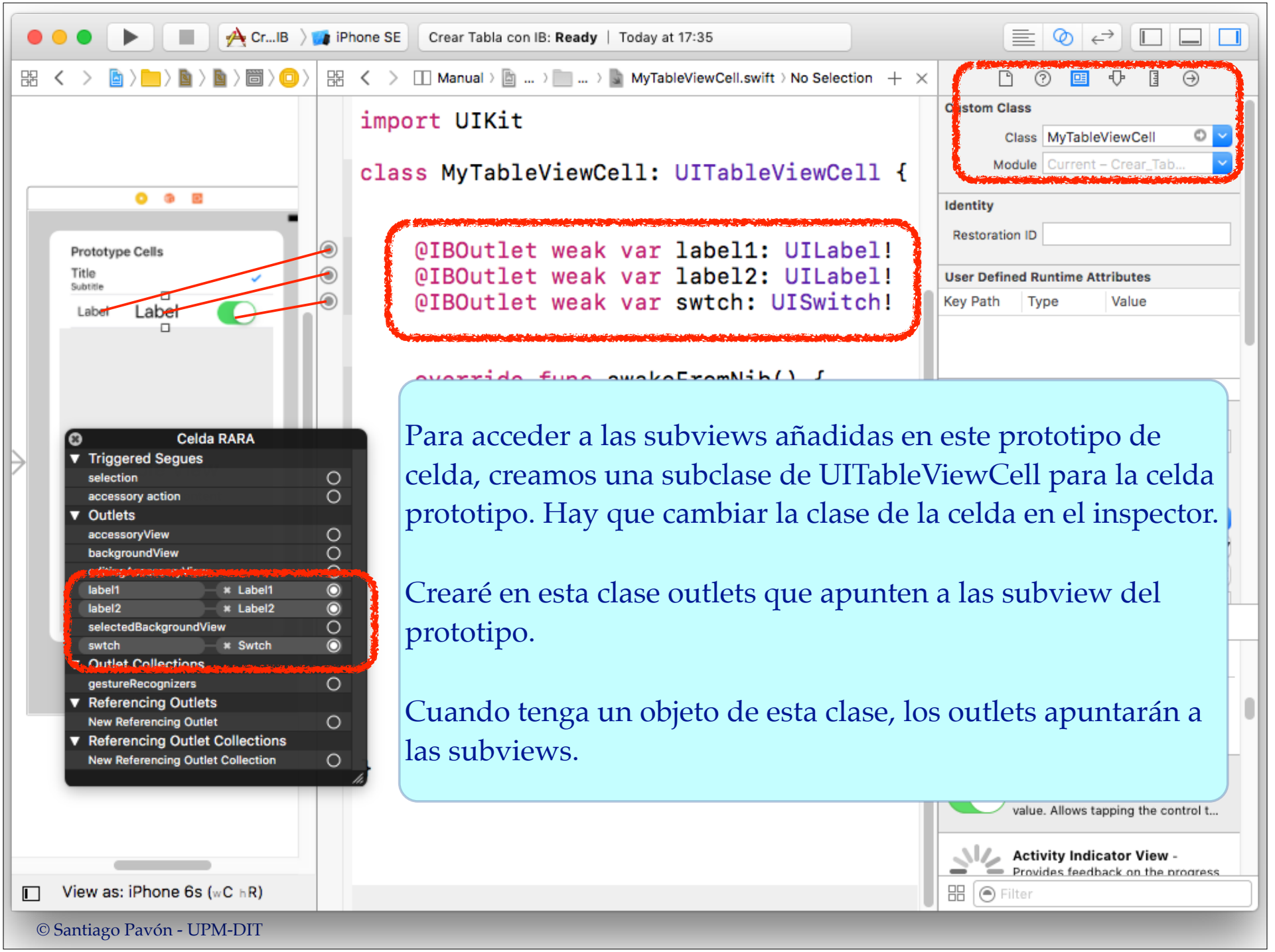

© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT

# 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.
    - . . .

© Santiago Pavón - UPM-DIT

# UITableViewDataSource

- **Configurar la Table View**

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

func tableView(_ tableView: UITableView,
     cellForRowAt indexPath: IndexPath) -> UITableViewCell

optional func numberOfSections(in tableView: UITableView) -> Int

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

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

© Santiago Pavón - UPM-DIT

- **Índice**

```
optional func sectionIndexTitles(for tableView: UITableView)
                                                -> [String]?

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

- **Añadir o borrar filas**

```
optional func tableView(_ tableView: UITableView,
              commit editingStyle: UITableViewCellEditingStyle,
              forRowAt indexPath: IndexPath)

optional func tableView(_ tableView: UITableView,
              canEditRowAt indexPath: IndexPath) -> Bool
```

- **Reordenar las filas**

```
optional func tableView(_ tableView: UITableView,
              canMoveRowAt indexPath: IndexPath) -> Bool

optional func tableView(_ tableView: UITableView,
              moveRowAt sourceIndexPath: IndexPath,
              to destinationIndexPath: IndexPath)
```

© Santiago Pavón - UPM-DIT

# IndexPath

- Las instancias de la estructura **`IndexPath`** 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`**).

© Santiago Pavón - UPM-DIT

# 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 dequeueReusableCell(withIdentifier identifier: String,
                         for indexPath: IndexPath) -> UITableViewCell
```

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

```
func dequeueReusableCell(withIdentifier identifier: String)
                                                  -> UITableViewCell?
```

  - 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: UITableViewCellStyle, reuseIdentifier: String?)
```

      - También existe un mecanismo para crear celdas nuevas para un identificador dado. Consiste en registrar en la tabla los ficheros NIB o crear las clases que se usarán para crear nuevas celdas.

```
func registerClass(_ cellClass: AnyClass?,
                   forCellReuseIdentifier identifier: String)
func registerNib(_ nib: UINib?,
                 forCellReuseIdentifier identifier: String)
```

© Santiago Pavón - UPM-DIT

Identificar tipo de celda o prototipo

```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath)
                        -> UITableViewCell {

  // Saca una celda de la cola para reutilizar
  let cell = tableView.dequeueReusableCell(withIdentifier: "My Cell Id",
                                           for: 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 = .checkmark

  // Devolver la celda
  return cell
}
```

© Santiago Pavón - UPM-DIT

# Cabeceras y Pies

- UITableViewDataSource declara métodos para obtener Strings que pueden usarse como cabecera y pie de las secciones de la tablas.
- También puede usarse UIViews
  - como cabeceras y pies de tabla.
    - Devueltas por las propiedades `tableHeaderView` y `tableFooterView`.
  - como cabeceras y pies de las secciones.
    - Devueltas por los métodos `headerView(forSection: Int)` y `footerView(forSection: Int)`.
- De manera similar a como se hace con las UITableViewCell,
  - Para crear estas cabeceras y pies pueden registrarse ficheros NIB o crear clases.

```
func registerNib(:,forHeaderFooterViewReuseIdentifier:)
func registerClass(:,forHeaderFooterViewReuseIdentifier:)
```

  - Y existe un mecanismo de reutilización de estas views, asociándolas a identificadores de reutilización.

```
func dequeueReusableHeaderFooterView(withIdentifier:)
                                -> UITableViewHeaderFooterView?
```

- *Nota sobre la edición de storyboards con IB:*
  - *También pueden arrastrarse Views desde la biblioteca de objetos al principio y final de las tablas para que actúen como cabeceras y pie de tabla.*

© Santiago Pavón - UPM-DIT

# UITableView - Actualizaciones

```
func reloadData()

func insertSections(_ sections: IndexSet,
                    with animation: UITableViewRowAnimation)

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

func reloadSections(_ sections: IndexSet,
                    with animation: UITableViewRowAnimation)

func insertRows(at indexPaths: [IndexPath],
                with animation: UITableViewRowAnimation)

func deleteRows(at indexPaths: [IndexPath],
                with animation: UITableViewRowAnimation)


func reloadRows(at indexPaths: [AnyObject],
                with animation: UITableViewRowAnimation)
. . .
```

© Santiago Pavón - UPM-DIT

# 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, ...

© Santiago Pavón - UPM-DIT

# UITableViewDelegate

- Configurar las filas de la tabla:

```
tableView(UITableView, heightForRowAt: IndexPath)
tableView(UITableView, estimatedHeightForRowAt: IndexPath)
tableView(UITableView, indentationLevelForRowAt: IndexPath)
tableView(UITableView, willDisplay: UITableViewCell,
          forRowAt: IndexPath)
```

- Manejar los accesorios:

```
tableView(UITableView, editActionsForRowAt: IndexPath)
tableView(UITableView, accessoryButtonTappedForRowWith: IndexPath)
```

- Manejar la selección de celdas:

```
tableView(UITableView, willSelectRowAt: IndexPath)
tableView(UITableView, didSelectRowAt: IndexPath)
tableView(UITableView, willDeselectRowAt: IndexPath)
tableView(UITableView, didDeselectRowAt: IndexPath)
```

© Santiago Pavón - UPM-DIT

- Modificar la cabecera y pie de las secciones:

```
tableView(UITableView, viewForHeaderInSection: Int)
tableView(UITableView, viewForFooterInSection: Int)
tableView(UITableView, heightForHeaderInSection: Int)
tableView(UITableView, estimatedHeightForHeaderInSection: Int)
tableView(UITableView, heightForFooterInSection: Int)
tableView(UITableView, estimatedHeightForFooterInSection: Int)
tableView(UITableView, willDisplayHeaderView: UIView, forSection: Int)
tableView(UITableView, willDisplayFooterView: UIView, forSection: Int)
```

- Edición de las filas de la tabla:

```
tableView(UITableView, willBeginEditingRowAt: IndexPath)
tableView(UITableView, didEndEditingRowAt: IndexPath?)
tableView(UITableView, editingStyleForRowAt: IndexPath)
tableView(UITableView,
          titleForDeleteConfirmationButtonForRowAt: IndexPath)
tableView(UITableView, shouldIndentWhileEditingRowAt: IndexPath)
```

- Reordenar las filas de la tabla:

```
tableView(UITableView, targetIndexPathForMoveFromRowAt: IndexPath,
          toProposedIndexPath: IndexPath)
```

© Santiago Pavón - UPM-DIT

- Seguimiento del borrado de views:

```
tableView(UITableView, didEndDisplaying: UITableViewCell,
          forRowAt: IndexPath)
tableView(UITableView, didEndDisplayingHeaderView: UIView, forSection: Int)
tableView(UITableView, didEndDisplayingFooterView: UIView, forSection: Int)
```

- Gestionar el copiado y pegado del contenido de las filas:

```
tableView(UITableView, shouldShowMenuForRowAt: IndexPath)
tableView(UITableView, canPerformAction: Selector, forRowAt: IndexPath,
          withSender: Any?)
tableView(UITableView, performAction: Selector, forRowAt: IndexPath,
          withSender: Any?)
```

- Gestionar el resaltado de las filas:

```
tableView(UITableView, shouldHighlightRowAt: IndexPath)
tableView(UITableView, didHighlightRowAt: IndexPath)
tableView(UITableView, didUnhighlightRowAt: IndexPath)
```

- Gestionar el foco:

```
tableView(UITableView, canFocusRowAt: IndexPath)
tableView(UITableView, shouldUpdateFocusIn: UITableViewFocusUpdateContext)
tableView(UITableView, didUpdateFocusIn: UITableViewFocusUpdateContext,
          with: UIFocusAnimationCoordinator)
indexPathForPreferredFocusedView(in: UITableView)
```

© Santiago Pavón - UPM-DIT

- Gestión de acciones Swipe: Nuevo en iOS 11

```
tableView(_: UITableView,
    leadingSwipeActionsConfigurationForRowAt: IndexPath)
      -> UISwipeActionsConfiguration?
tableView(_: UITableView,
    trailingSwipeActionsConfigurationForRowAt indexPath: IndexPath)
      -> UISwipeActionsConfiguration?
```

- etc...

© Santiago Pavón - UPM-DIT

# 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,
               didSelectRowAt indexPath: IndexPath)
```

    - 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 `prepare(for segue:, sender:)` para:
    - Consultar que celda está seleccionada para saber dónde hemos pulsado.
    - Configurar el VC destino del segue.

© Santiago Pavón - UPM-DIT

# Ejemplo

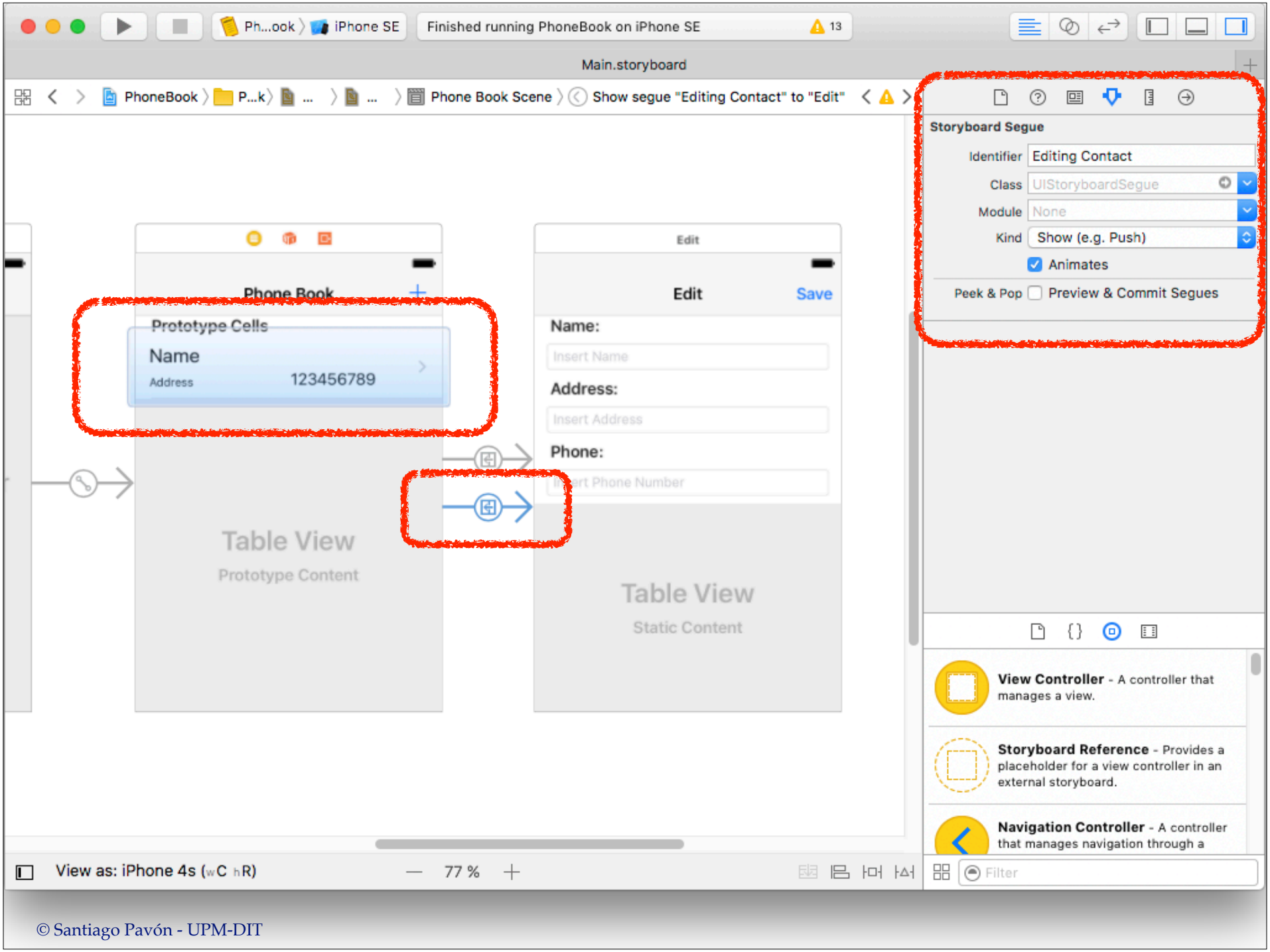
```
override func tableView(_ tableView: UITableView,
                        didSelectRowAt indexPath: IndexPath) {

  // Creo un objeto VC nuevo cargandolo del storyboard
  if let ivc = storyboard?.instantiateViewController(withIdentifier: "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)
  }
}
```

© Santiago Pavón - UPM-DIT

# Ejemplo

```
override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
  if segue.identifier == "Editing Contact" {
    if let ecvc = segue.destination as? EditContactTableVC,
      let ip = tableView.indexPathForSelectedRow {
          ecvc.contact = contactBook.contacts[ip.row]
    }
  } else if segue.identifier == "Adding Contact" {
    if let ecvc = segue.destination as? EditContactTableVC {
        let newContact = Contact(name: "", address: "", phone: "")
        contactBook.contacts.insert(newContact, at: 0)
        let ip = IndexPath(row: 0, section: 0)
        tableView.insertRows(at: [ip], with: .automatic)
        tableView.selectRow(at: ip, animated: true, scrollPosition: .top)
        ecvc.contact = newContact
    }
  }
}
```

© Santiago Pavón - UPM-DIT



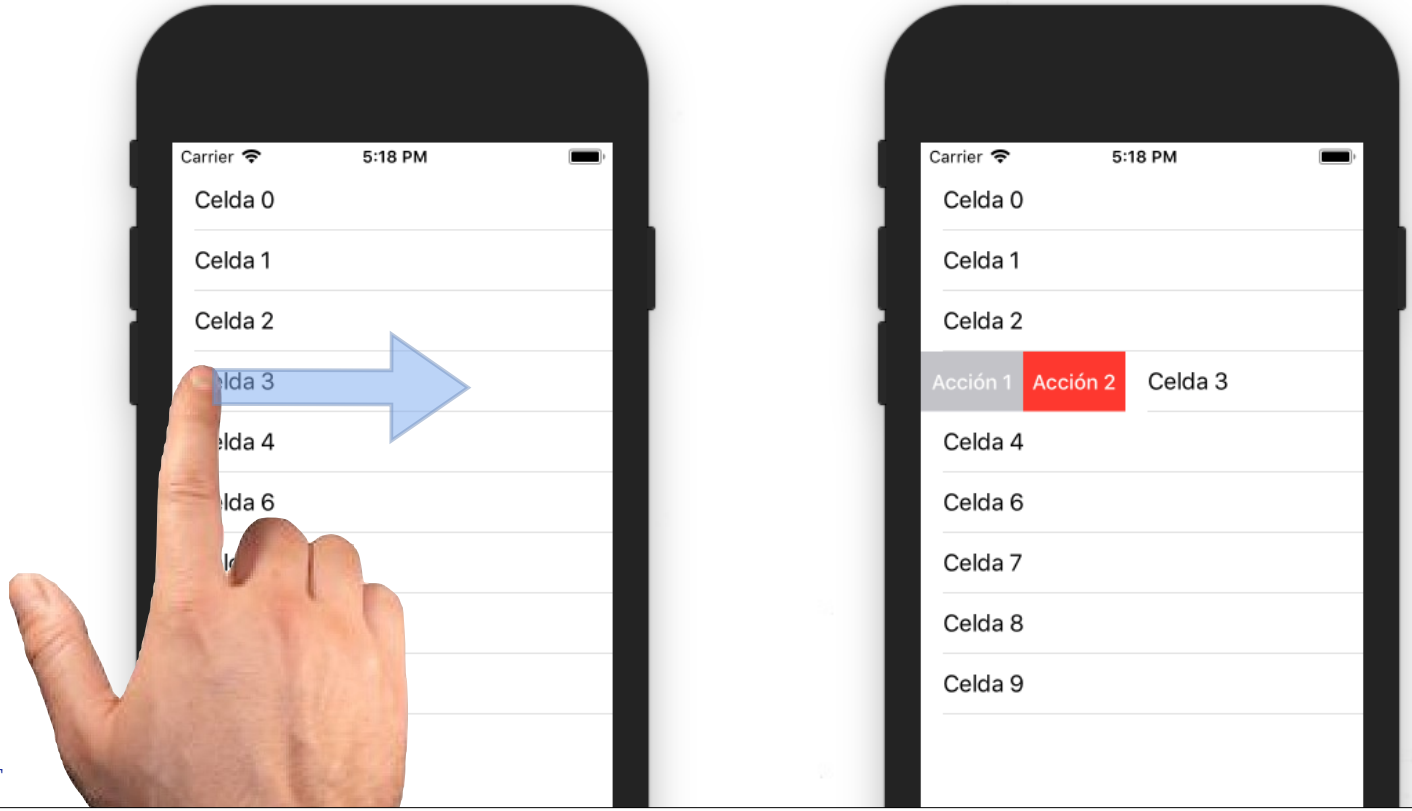

© Santiago Pavón - UPM-DIT

# Ejemplo: Action Swipe

- Nuevo en iOS 11.
- Programamos en el delegado de la tabla el método

  `tableView(:, leadingSwipeActionsConfigurationForRowAt:)`

  - para indicar que acciones queremos ejecutar al hacer swipe desde el lado leading.

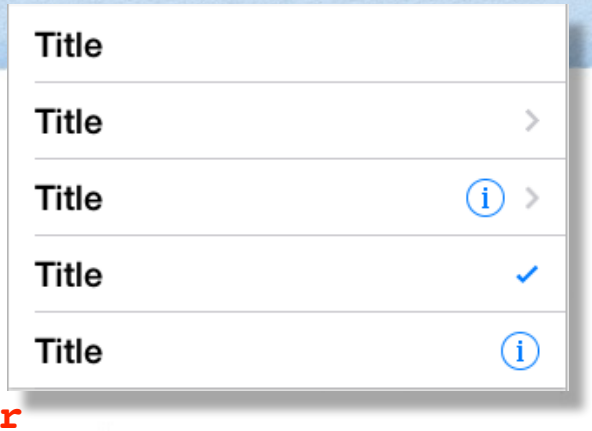



© Santiago Pavón - UPM-DIT

```
override func tableView(_ tableView: UITableView,
        leadingSwipeActionsConfigurationForRowAt indexPath: IndexPath)
                -> UISwipeActionsConfiguration? {

    // Una accion: su estilo, el titulo y lo que hay que hacer
    let ca1 = UIContextualAction(style: .normal, title: "Acción 1") {
        action, sourceView, completionHandler in

            print("Realizada Accion 1")
            completionHandler(true)  // Digo que si he hecho la accion
        }

    // Otra accion: su estilo, el titulo y lo que hay que hacer
    let ca2 = UIContextualAction(style: .destructive, title: "Acción 2") {
        action, sourceView, completionHandler in

           print("Realizada Accion 2")
           completionHandler(true)  // Digo que si he hecho la accion
        }

    // Configuracion de todas las acciones:
    let c = UISwipeActionsConfiguration(actions: [ca1, ca2])
    c.performsFirstActionWithFullSwipe = true  // swipe a lo bestia

    return c
}
```

© Santiago Pavón - UPM-DIT

# 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,
               accessoryButtonTappedForRowWith indexPath: IndexPath)
```

- 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.

© Santiago Pavón - UPM-DIT

# Tamaño de las Celdas

- La altura con la que pintan **todas** las celdas puede:
  - Configurarse desde el inspector del **Interface Builder**.
  - o indicarse **programáticamente** asignando un valor a la propiedad `tableView.rowHeight`.
  - Todas las celdas se pintarán con la altura configurada, independientemente de cual sea la altura necesaria para pintar su contenido.

© Santiago Pavón - UPM-DIT

- Si cada celda tiene una altura distinta, se puede indicar cual es la altura de cada celda sobrescribiendo el método:

```
func tableView(_ tableView: UITableView,
               heightForRowAt indexPath: IndexPath)
                                          -> CGFloat
```

  - 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 sobrescribir en el delegado de la tabla el método:

```
func tableView(_ tableView: UITableView,
               estimatedHeightForRowAt indexPath: IndexPath)
                                          -> CGFloat
```

© Santiago Pavón - UPM-DIT

- El cálculo del tamaño individual de cada celda puede hacerse automáticamente usando autolayout.
  - Hay que asegurarse de que las restricciones de autolayout añadidas a la celda (en su prototipo) determinan cuál es su altura.
    - Las restricciones añadidas junto con el tamaño intrínseco de las subviews de cada celda determinan la altura de la celda.
    - El ancho lo determina la tableView.
  - 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
      ```

  - De esta forma, 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.

© Santiago Pavón - UPM-DIT

# Editar una Tabla

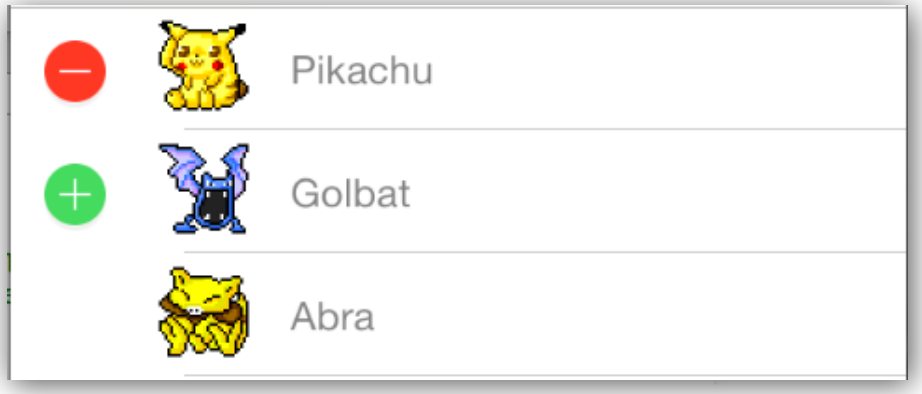
- Preguntar al **data source** si la celda es editable.

```
func tableView(_ tableView: UITableView,
     canEditRowAt indexPath: IndexPath) -> Bool
```

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

```
func tableView(_ tableView: UITableView,
     editingStyleForRowAt indexPath: IndexPath)
     -> UITableViewCellEditingStyle
```

  - Se muestra un icono indicando el tipo de edición




© Santiago Pavón - UPM-DIT

- 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,
               commit editingStyle: UITableViewCellEditingStyle,
               forRowAt indexPath: IndexPath)
```

  - 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()
insertRows(at indexPaths:, with rowAnimation:)
deleteRows(at indexPaths:, with rowAnimation:)
```

© Santiago Pavón - UPM-DIT

- 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:

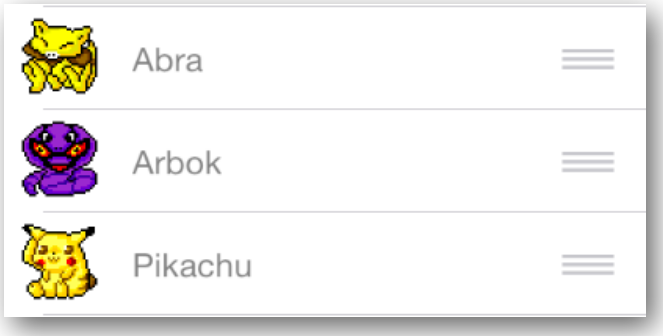
```
func tableView(_ tableView: UITableView,
               canMoveRowAtIndexPath indexPath: IndexPath)
                  -> Bool
```

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

```
func tableView(_ tableView: UITableView,
                  moveRowAt sourceIndexPath: IndexPath,
                  to destinationIndexPath: IndexPath)
```

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

```
var showsReorderControl: Bool
```




© Santiago Pavón - UPM-DIT

# 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.

© Santiago Pavón - UPM-DIT

# `UITableViewController` - Modo Edición

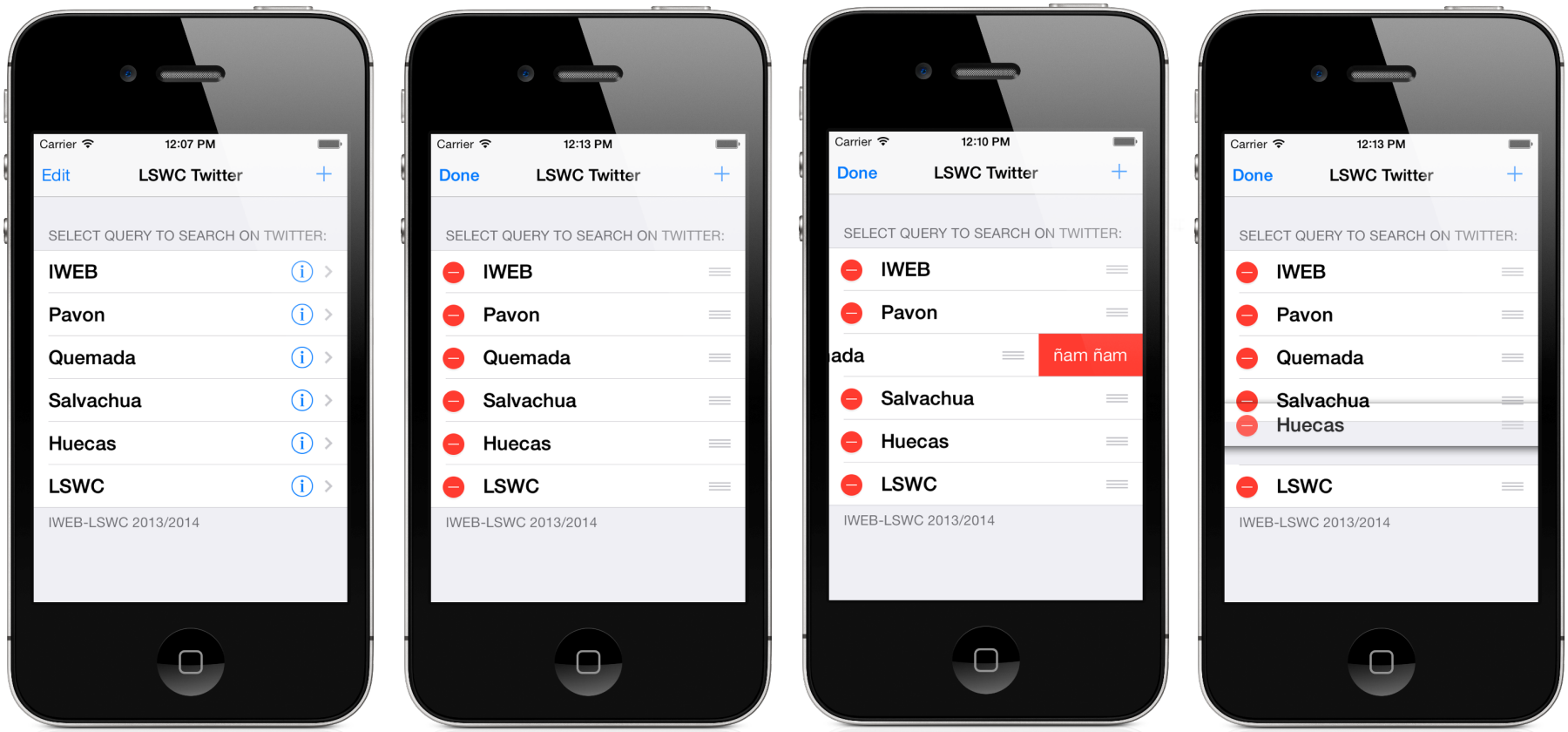
- Los objetos **UIViewController** tienen una propiedad booleana (**`isEditing`**) que indican si estoy editando el contenido del VC.
- También proporcionan una propiedad (**`editButtonItem`**) que apunta a un botón que podemos colocar en la barra de navegación.
  - Este botón nos permite activar y desactivar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

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

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

© Santiago Pavón - UPM-DIT

# Ejemplo: editar una tabla




© Santiago Pavón - UPM-DIT

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

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

    loadQueries()
}


override func tableView(_ tableView: UITableView,
              titleForDeleteConfirmationButtonForRowAt
                      indexPath: IndexPath) -> String! {
    return "ñam ñam"
}
```

© Santiago Pavón - UPM-DIT

```
override func tableView(_ tableView: UITableView,
                        commit editingStyle: UITableViewCellEditingStyle,
                        forRowAt indexPath: IndexPath) {

    if editingStyle == .delete {

        // Borro el dato de esa fila de mi modelo.
        queries.remove(at: indexPath.row)

        // Actualizar lo que muestra la table view
        tableView.deleteRows(at: [indexPath],
                             with: .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.
    }

}
```

© Santiago Pavón - UPM-DIT

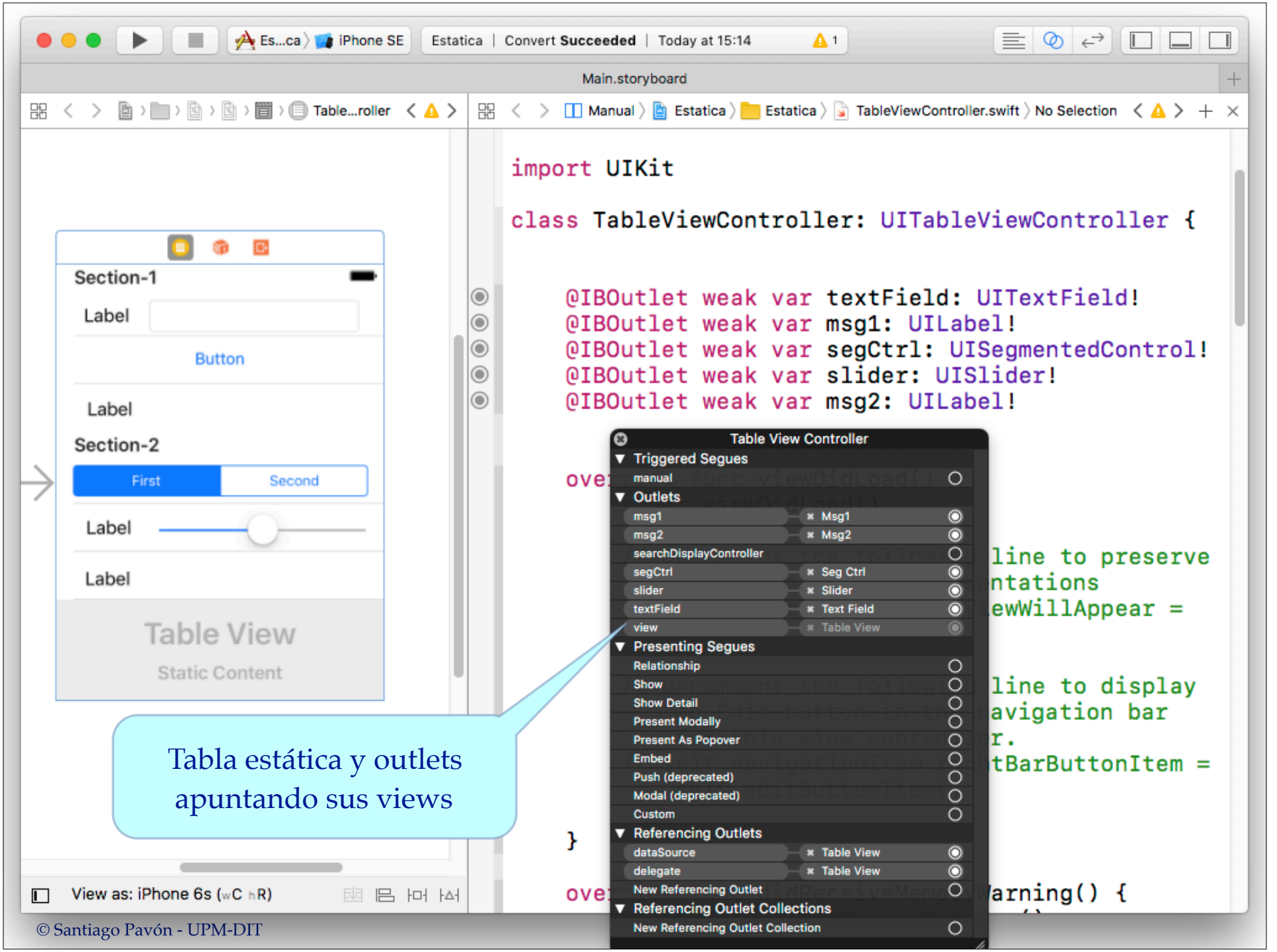
```
override func tableView(_ tableView: UITableView,
                        moveRowAt fromIndexPath: IndexPath,
                        to: IndexPath) {

    let obj = queries[fromIndexPath.row]

    queries.remove(at: fromIndexPath.row)
    queries.insert(obj, at: indexPath.row)

    saveQueries()
}


override func tableView(_ tableView: UITableView,
                        canMoveRowAt indexPath: IndexPath) -> Bool {
    return true
}
```

© Santiago Pavón - UPM-DIT

# 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.
- Para acceder a las subviews añadidas a las celdas, tanto en celdas estáticas y prototipos de estilo personalizado, se recomendable:
  - crear nuestras propias subclases de celda, derivadas de UITableViewCell, y crear outlets apuntando a las subviews.

© Santiago Pavón - UPM-DIT



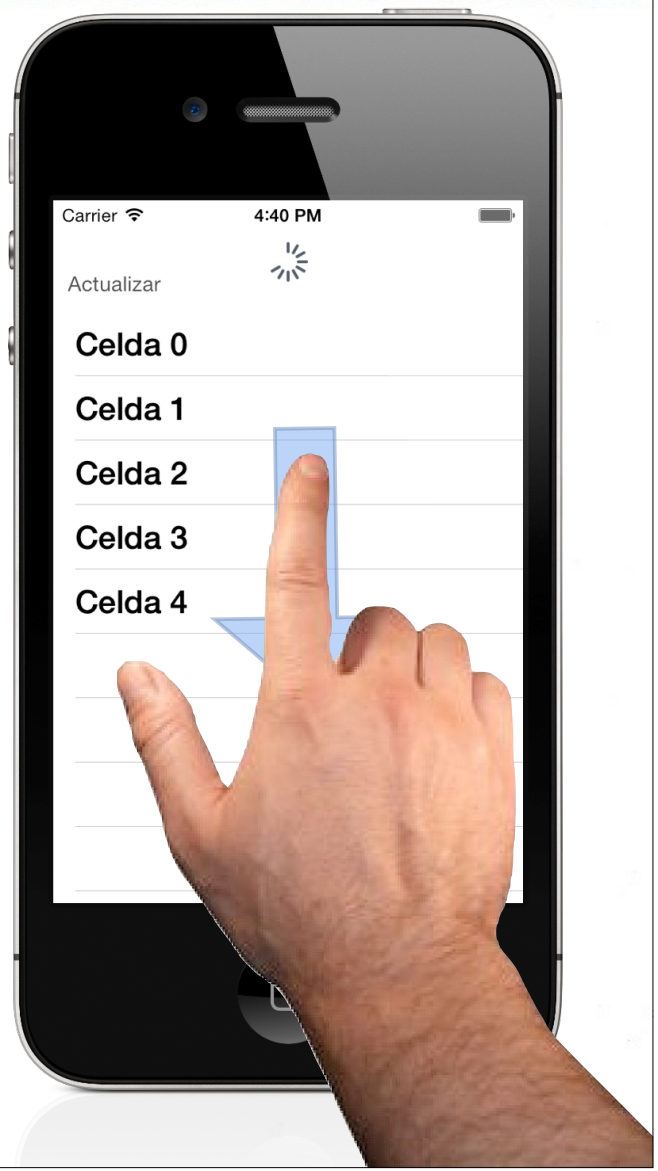

© Santiago Pavón - UPM-DIT

# Estirar para Refrescar

© Santiago Pavón - UPM-DIT

# 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




© Santiago Pavón - UPM-DIT

# Programáticamente

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

    refreshControl = UIRefreshControl()

    refreshControl?.addTarget(self,
                              action: #selector(refreshMyTable),
                              for: .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.

© Santiago Pavón - UPM-DIT

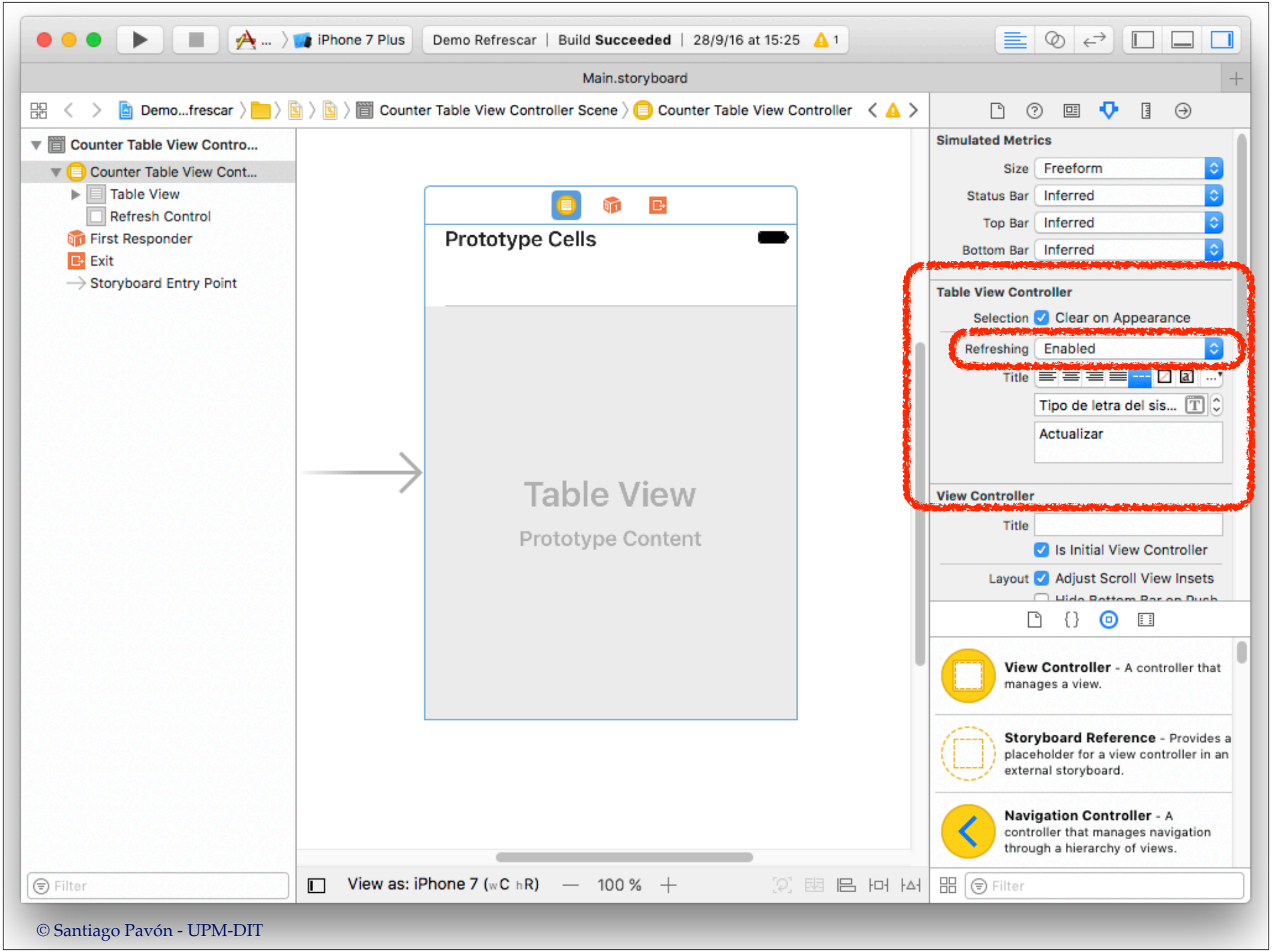
# Con Interface Builder

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

    // refreshControl = UIRefreshControl()

    // refreshControl?.addTarget(self,
    //                           action: #selector(refreshMyTable),
    //                           for: .valueChanged)
}


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

   // Recargar tabla
   tableView.reloadData()

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

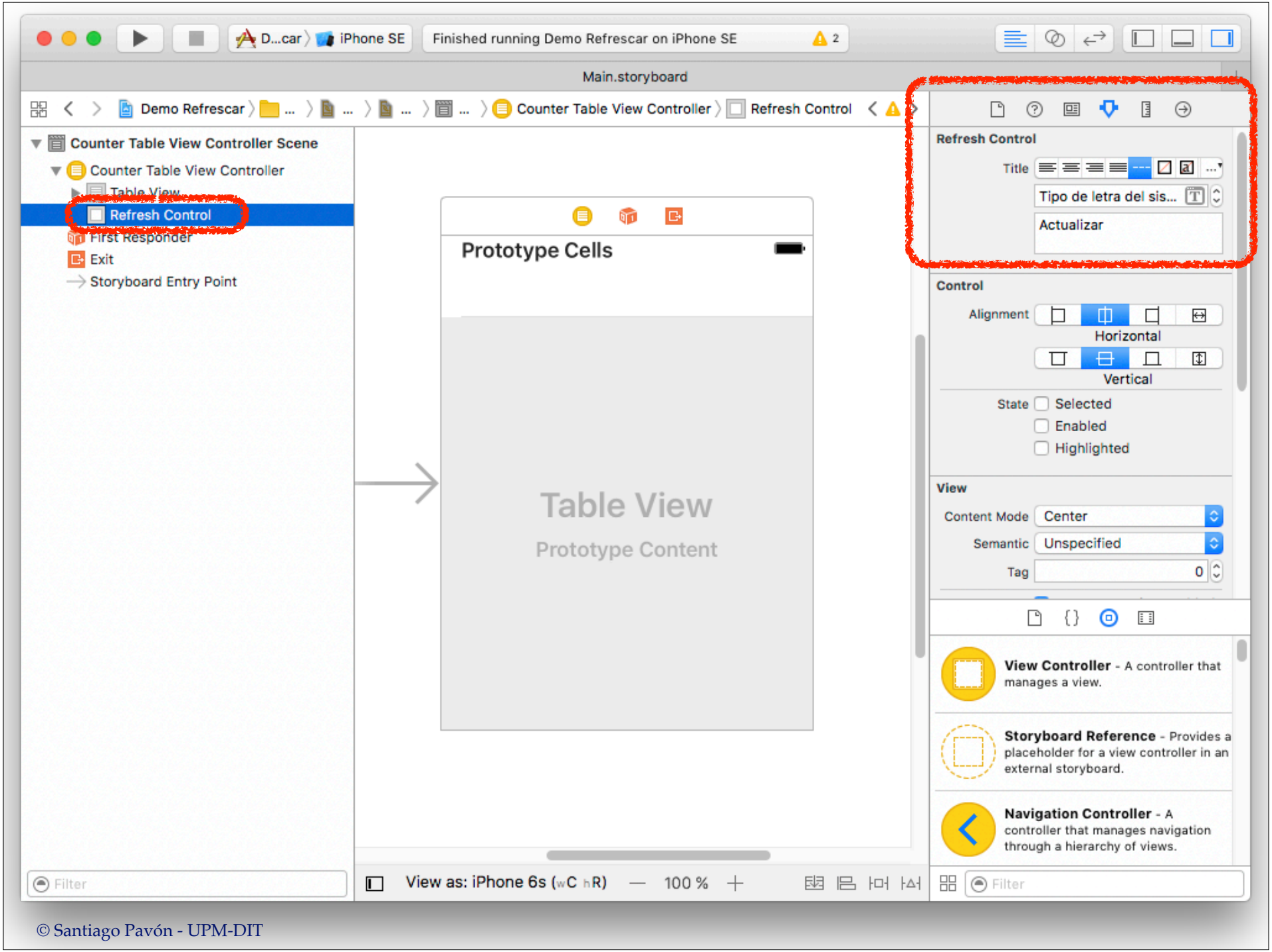
El objeto **self.refreshControl** lo creo con IB

La acción la creo como una IBAction con IB

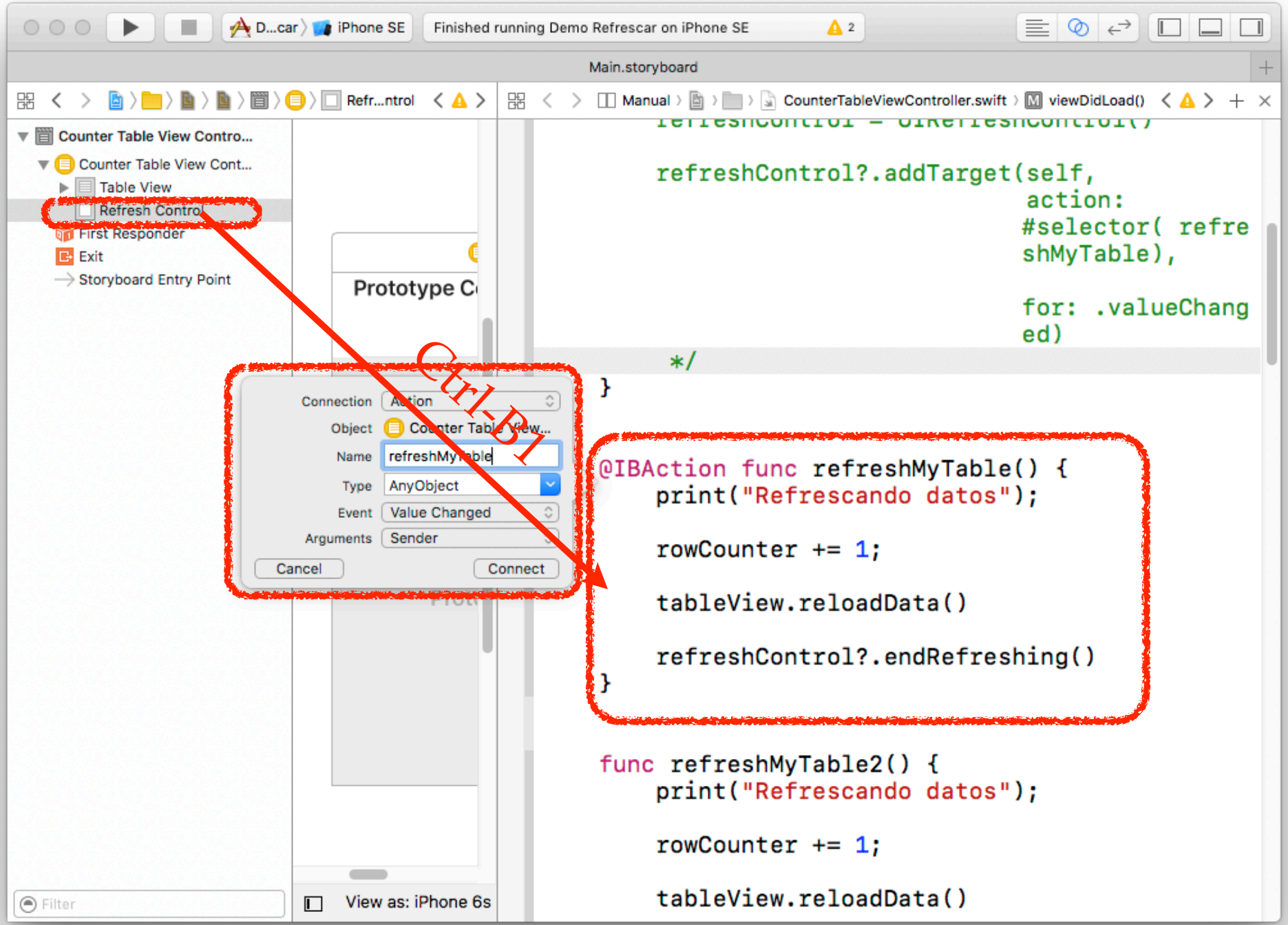
© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT




© Santiago Pavón - UPM-DIT

© Santiago Pavón - UPM-DIT