LÍNEAS DE PRODUCTOS SOFTWARE

Sesión 04. XCode, Swift, iOS: Tablas y navegación

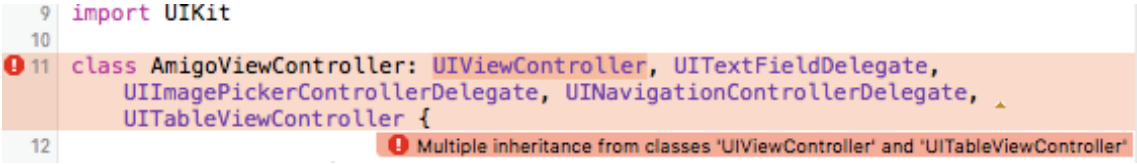
Aitor García Luiz
Grado en Ingeniería Informática

La sesión 04 de prácticas parte de lo realizado anteriormente en la sesión 03. Antes de comenzar los ejercicios se deberá extender la misma siguiendo los pasos del guion de prácticas.

Ejercicio 1.

¿Sería posible reutilizar nuestro controlador de vista ya existente? (ViewController.swift)

Vamos a probar que pasaría si usamos el mismo controlador que ya teníamos:



Como podemos observar se produce un error que nos indica que no es posible realizar una herencia múltiple desde las clases UIViewController y UITableViewController, es por esto, que necesitamos un nuevo controlador.

Ejercicio 2.

¿Qué modificación habría que hacer para que la siguiente línea de código no diera lugar a un error? ¿Por qué?

```
amigos += [amigo1, amigo2, amigo3]
```

Sería necesario añadir un símbolo de exclamación "!" al final de cada variable, para forzar a que su valor tenga que ser válido (en este caso de tipo Amigo) y no pueda ser nil o de otro tipo.

Ejercicio 3.

Investigad las propiedades de tabla, celda… y comprobad el resultado de su modificación. Por ejemplo, ¿qué atributo es el que añade el símbolo > al final de cada celda? ¿Cuántos valores posibles puede tener este atributo? ¿Cuál es su resultado en la interfaz?

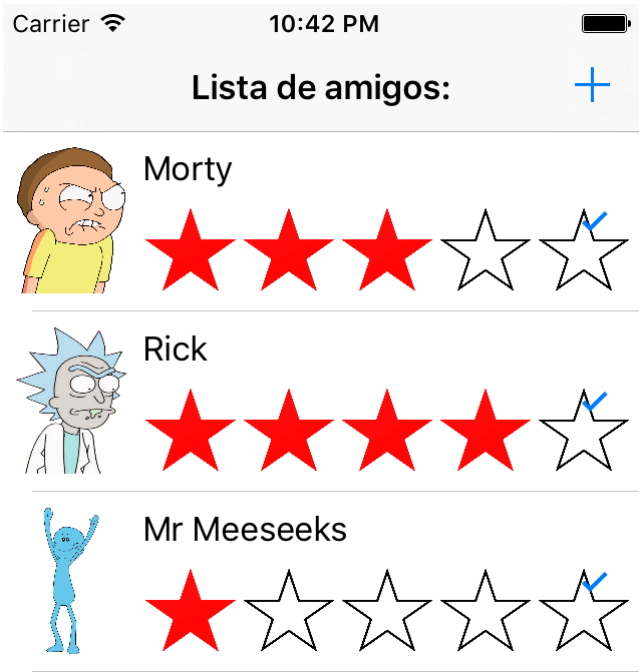
Gracias a la propiedad *accesoryType* de la celda, podemos obtener las diferentes modificaciones:

```
override func tableView(tableView: UITableView, cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
    let cell = tableView.dequeueReusableCellWithIdentifier("AmigoTableViewCell", forIndexPath: indexPath) as!
        AmigoTableViewCell
    cell.nombre.text = amigos[indexPath.row].nombre
    cell.foto.image = amigos[indexPath.row].foto
    cell.evaluacion.gradoAfinidad = amigos[indexPath.row].gradoAfinidad
    cell.accessoryType = .
```

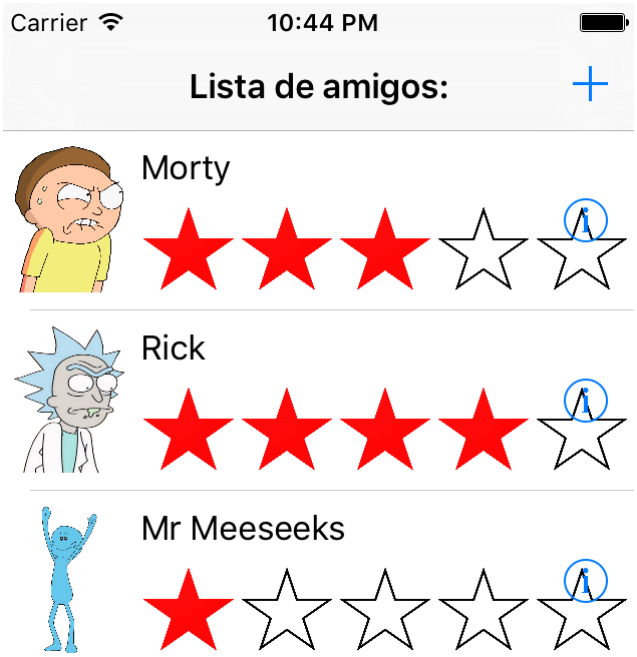
```
UITableViewCellAccessoryType Checkmark
UITableViewCellAccessoryType DetailButton
UITableViewCellAccessoryType DetailDisclosureButton
UITableViewCellAccessoryType DisclosureIndicator
UITableViewCellAccessoryType None
```

Como podemos observar hay 5 opciones diferentes:

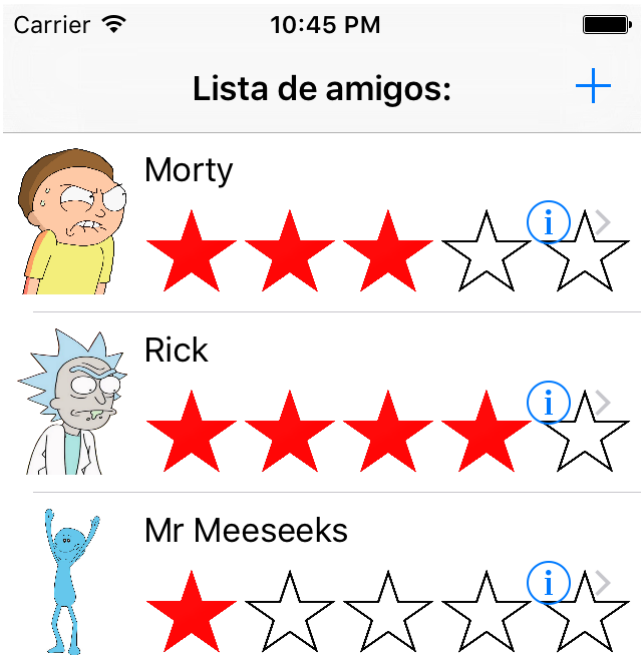
1. **Checkmark:**



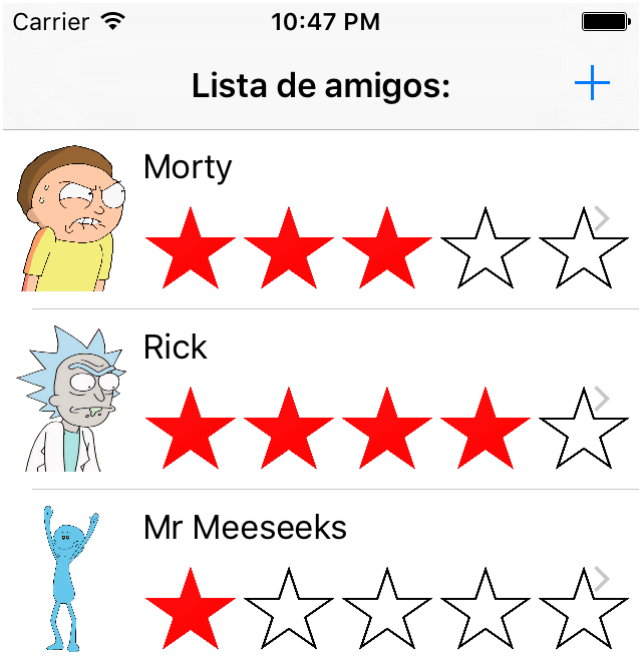
2. DetailButton:



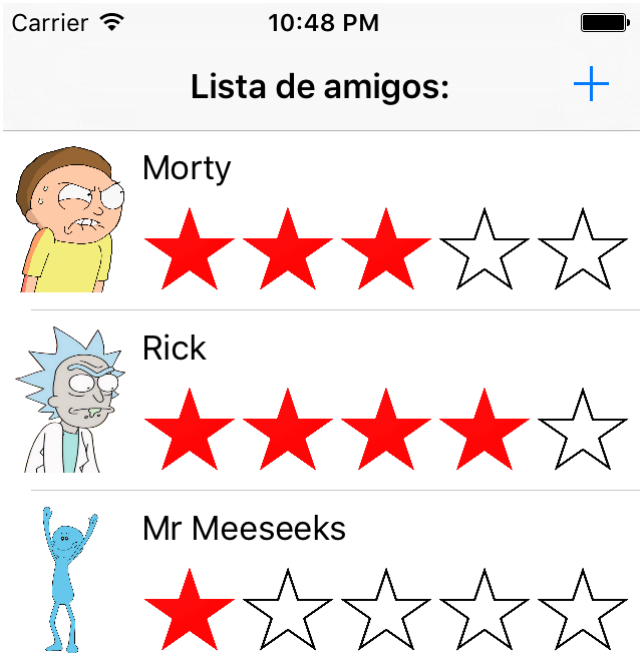
3. DetailDisclosureButton:



4. **DisclosureIndicator:**



5. **None:**



Ejercicio 4.

Explica detalladamente en qué consiste este error fatal.

Si no se especifica un nombre para el nuevo amigo ocurrirá el siguiente error:

```
// MARK: - Unwind segue desde AmigoViewController

@IBAction func addNuevoAmigo(sender: UIStoryboardSegue){
    let sourceViewController = sender.sourceViewController as! AmigoViewController
    let nuevoAmigo = sourceViewController.amigo
    amigos.append(nuevoAmigo!)                    Thread 1: EXC_BAD_INSTRUCTION (code=EXC_I386_INVOP, subcode=0x0)
    let newIndexPath = NSIndexPath(forRow: amigos.count-1, inSection: 0)
    tableView.insertRowsAtIndexPaths([newIndexPath], withRowAnimation: .Bottom)
}
```

¿Qué es lo que ha ocurrido? Es bastante sencillo. En la línea anterior a la marcada se crea una constante 'nuevoAmigo' que llamará al constructor de Amigo.swift:

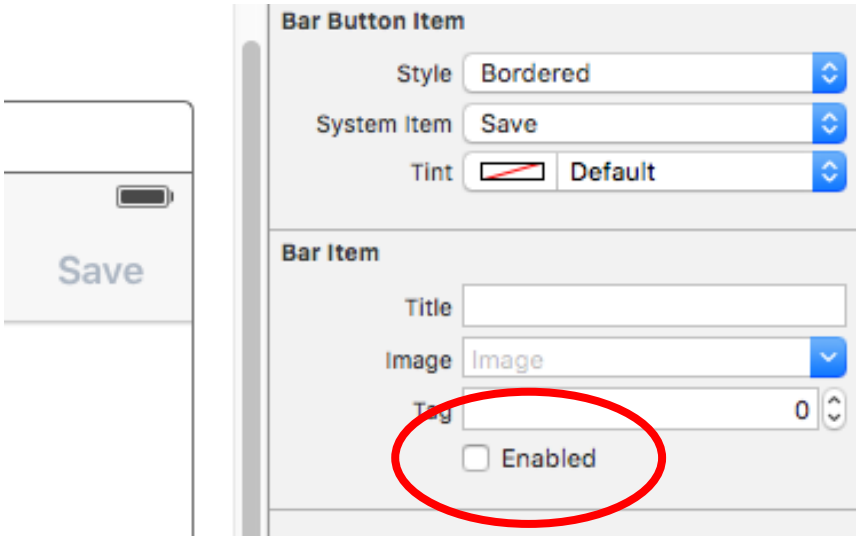
```
//MARK: Inicialización
init?(nombre: String, foto: UIImage?, gAfinidad: Int){
    if nombre.isEmpty || gAfinidad < 0 {
        return nil
    }

    self.nombre = nombre
    self.foto = foto
    self.gradoAfinidad = gAfinidad

}
```

Como podemos ver en su código, la sentencia if comprueba si el nombre está vacío (o si el grado de afinidad es menor de 0) y devuelve el valor nil en caso de que ocurra. Por lo tanto, al introducir nuestro nuevo amigo con valor nil y realizarle un 'wrapp' con el signo de exclamación "!" que no permite ese valor, pues se provoca dicho error fatal.

Ejercicio 5.

Como solución, vamos a escribir el código necesario que asegure que siempre vamos a tener un valor no vacío en el campo de texto. Para ello, vamos a jugar con el atributo *enabled* del botón *Save*, activando o desactivando dependiendo del valor del campo de texto.

Para desactivar el botón podemos llamar a su propiedad *enabled* y poner su valor a *false*, o hacerlo mediante la interfaz gráfica, para ello hay que seleccionar el botón Save y en el Inspector de Atributos tenemos que desmarcar la casilla de enabled.



Una vez puesto su estado por defecto en false, solo queda comprobar mediante una condición if en qué momentos se activará y cuándo volverá a desactivarse:

```
func textFieldDidEndEditing(textField: UITextField) {
    if(nombreTF.text != ""){
        saveBtn.enabled = true
    } else {
        saveBtn.enabled = false
    }
}
```

Es decir, el botón solo se activará si hemos finalizado la acción de introducir una cadena en el campo de texto. Si después de haberse activado dicho botón, borramos el contenido del campo de texto, el botón volverá a desactivarse.

Capturas de ejemplo del funcionamiento del botón: