PASO A PASO 4:
UITableViewController - Cell Favorite

Vamos a crear una aplicación conformada por una tabla, que va a listar información de contactos. Además, agregaremos un mecanismo que nos permita marcar un contacto como favorito desde la celda correspondiente en la tabla.
Nota: este concepto aplica a cualquier tipo de acción que se ejecute desde una celda.

1. Tomaremos como punto de partida la aplicación armada en *Paso a Paso 1*. La misma consta de ***ContactsTableViewController***, un `UITableViewController`, que lista los contactos del usuario; ***ContactTableViewCell***, una `UITableViewCell` custom; y ***Person***, una clase que representa a un contacto de la Agenda.

2. En principio vamos a agregar un botón a la celda, que va a permitir marcar el contacto como favorito. Se crea también el Outlet correspondiente, y el Action asociado al evento 'TouchUpInside', que se llamará "favoriteTap". Como vamos a poder marcar un contacto como favorito, agregamos a la clase ***Person*** el atributo 'isFavorite". De este modo, cuando el usuario marque un contacto como favorito, se va a cambiar el estado del atributo.

3. El objetivo de agregar un botón es que podamos cambiar fácilmente el estado. Si el contacto es favorito el botón debería mostrar cierto contenido (por ejemplo una estrella amarilla rellena), mientras que si el contacto no es favorito el botón debería mostrar otro contenido (una estrella gris vacía). Esto se puede configurar fácilmente si usamos los distintos estados de un `UIButton`.

    3.1. En primera instancia agregaremos dos recursos al proyecto: una estrella rellena y una estrella vacía.

    3.2. Luego, en el storyboard, seleccionamos el botón y en el panel de Attributes Inspector vemos que el "state config" indica "default". Es decir, toda la configuración que hagamos del botón va a aplicar al estado default. Si cambiamos "state config" a "selected" vamos a poder configurar el botón para ese nuevo estado. Vamos entonces a asignar la imagen de estrella vacía al estado default; y la imagen de estrella rellena al estado selected.

    3.3. Finalmente, en la clase ***ContactTableViewCell***, en el método "setup", además de configurar los labels vamos a tener que configurar el botón. Si el

contacto es favorito, el botón debería mostrarse acorde a la configuración del estado selected; caso contrario, el botón debería mostrar según la configuración del estado default:

```
if person.isFavorite {
      favoriteButton.isSelected = true
} else {
      favoriteButton.isSelected = false
}
```

4. Cuando el usuario toca el botón de favorito (y se ejecuta el método "favoriteTap") queremos cambiar el estado del contacto asociado. Sin embargo, modificar el estado interno de un elemento es mucha responsabilidad para una tabla, demasiada responsabilidad. La celda sólo debe preocuparse de mostrar información, y cualquier otra cosa más allá de eso debería ser delegada al TableViewController que maneja la tabla.

   4.1. Así como previamente creamos un protocolo para informar al TableViewController que hay que agregar un nuevo elemento, vamos a crear un protocolo para informar que hay que modificar el estado interno de un elemento. Este protocolo será el ***ContactTableViewCellDelegate*** y lo definimos en el mismo archivo que la ***ContactTableViewCell***.

   4.2. El protocolo definirá un único método para informar que se actualizó un contacto: `didUpdateFavorite(contact: Person)`.

   4.3. Luego queremos que, al tocar el botón, la celda informe a su delegado del cambio. Para ello, primero debemos tener una referencia del delegado al que queremos pasarle la responsabilidad. Por lo tanto, agregaremos un atributo llamado "delegate" de tipo ***ContactTableViewCellDelegate?*** (opcional porque no va a estar definido hasta que se setee). Este atributo será seteado en el "setup", así que agregaremos un parámetro al método "setup" que sea `delegate: ContactTableViewCellDelegate`. Finalmente, dentro del método "setup" asignaremos el atributo con el parámetro recibido: `self.delegate = delegate`.

   4.4. Además de setear el "delegate", vamos a tener que mantener una referencia a la persona recibida para poder enviarsela al delegado; es decir, vamos a definir un atributo de tipo ***Person?*** (opcional porque inicialmente no va a estar definido) y en el método "setup" vamos a asignarle el objeto "person" recibido como parámetro.

   4.5. Finalmente, dentro del método "favoriteTap" vamos informarle al delegado:

```
if let thePerson = person {
        delegate?.didUpdateFavorite(contact: person)
}
```

5. Ahora, como queremos que nuestra tabla esté "observando" los cambios de la celda, necesitamos que ***ContactsTableViewController*** sea el delegado de la celda.

    5.1. Para ello, al configurar cada celda en el método "cellForRowAtIndexPath", cuando invocamos al setup deberemos agregar el nuevo parámetro "delegate" y el valor enviado va a ser el controller mismo, es decir "self".

    5.2. Además, debemos indicar que el controller implementa el nuevo protocolo ***ContactTableViewCellDelegate***; por lo cual debemos agregar la implementación del método "didUpdateFavorite".

    5.3. Dentro del método "didUpdateFavorite" simplemente vamos a modificar el estado interno del contacto, y luego recargar la tabla:

```
contact.isFavorite = !contact.isFavorite
tableView.reloadData()
```

ALTERNATIVA 2

6. En lugar de recargar toda la tabla cada vez que un elemento es modificado, se puede hacer que sólo se recargue la información modificada.

    6.1. De esta forma, el método "didUpdateFavorite" sólo debe modificar el estado interno del elemento, por lo que no se necesita recargar la tabla.

    6.2. Como no se va a recargar la tabla, sólo vamos a tener que modificar el estado del botón cuando el usuario lo toca. Es decir, dentro del método "favoriteTap", además de llamar al delegado, podemos modificar el estado del botón