UITableView

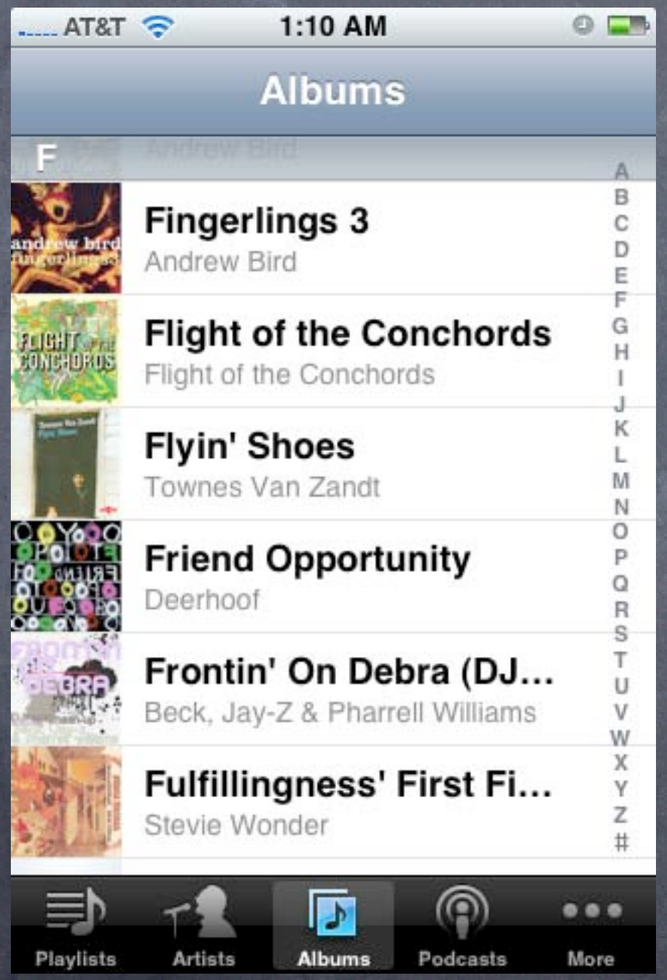
Matthias Tretter, @myell0w

UITableView

- Very important class for displaying data in a table
  - One-dimensional table.
  - It's a subclass of `UIScrollView`.
  - Table can be static or dynamic (i.e. a list of items).
  - Lots and lots of customization via a `dataSource` protocol and a `delegate` protocol.
  - Very efficient even with very large sets of data.
- Displaying multi-dimensional tables ...
  - Usually done via a `UINavigationController` containing multiple MVC's where View is `UITableView`
- Kinds of `UITableViews`
  - Plain or Grouped
  - Static or Dynamic
  - Divided into sections or not
  - Different formats for each row in the table (including completely customized)

UITableView

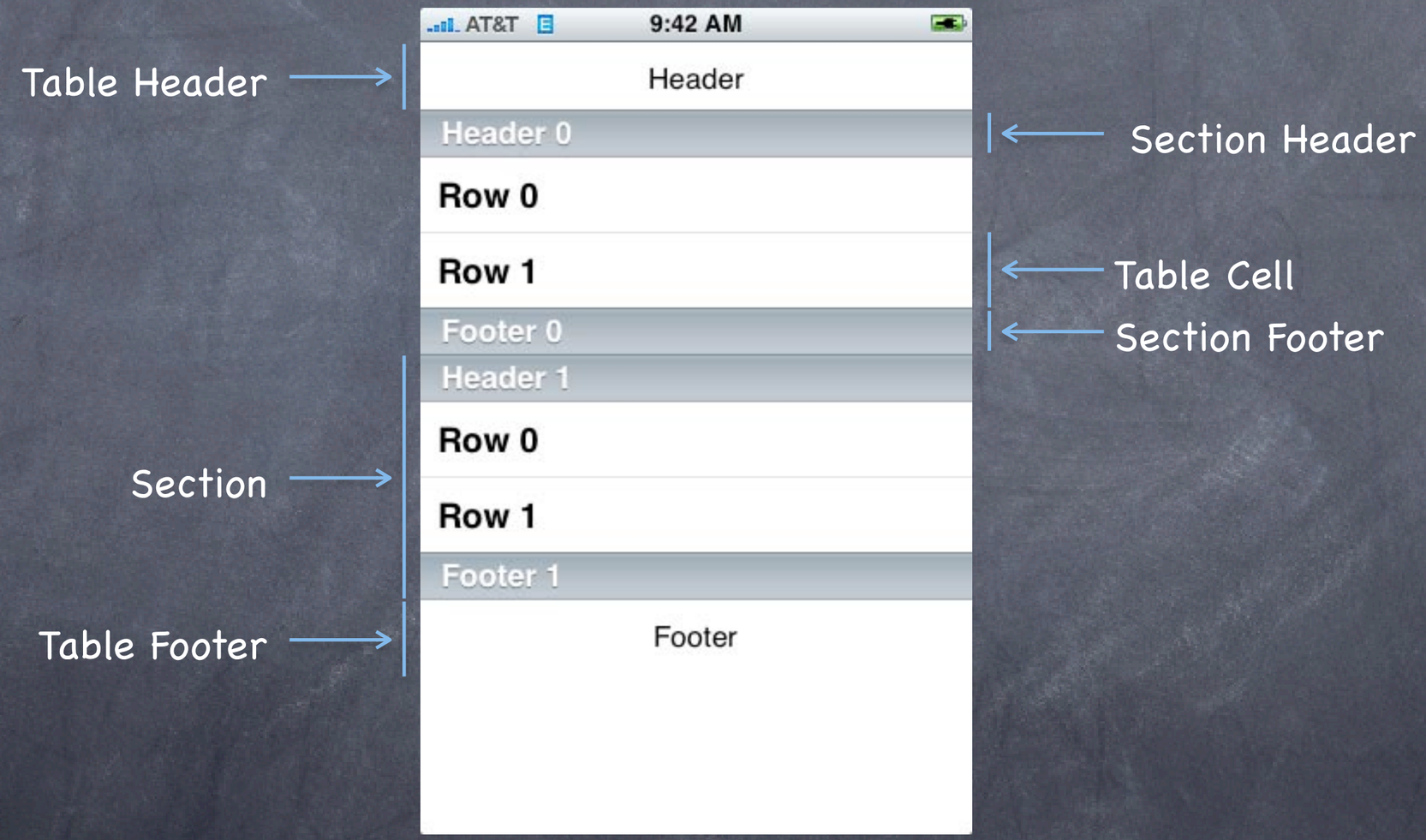
UITableViewStylePlain



UITableViewStyleGrouped

UITableView

Plain Style



UITableView

Grouped Style

Table Header

Section

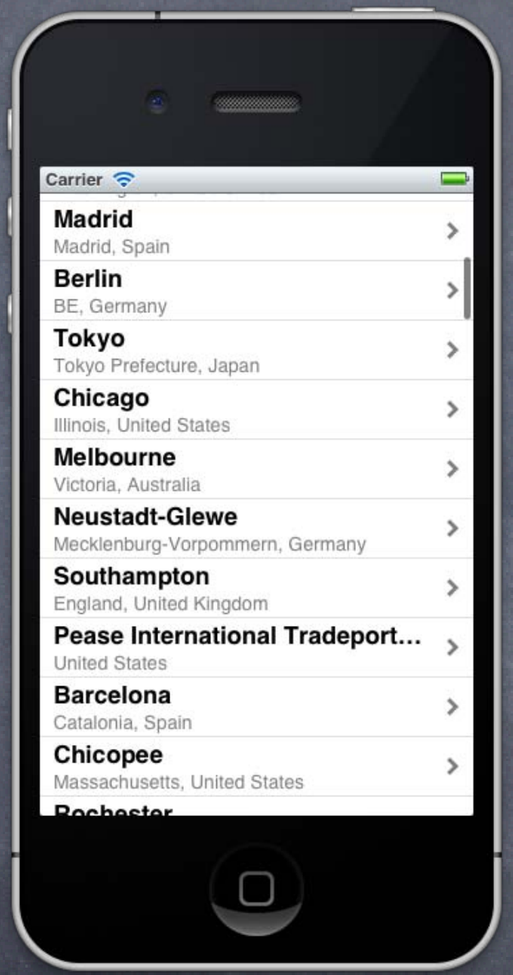
Table Footer

Section Header

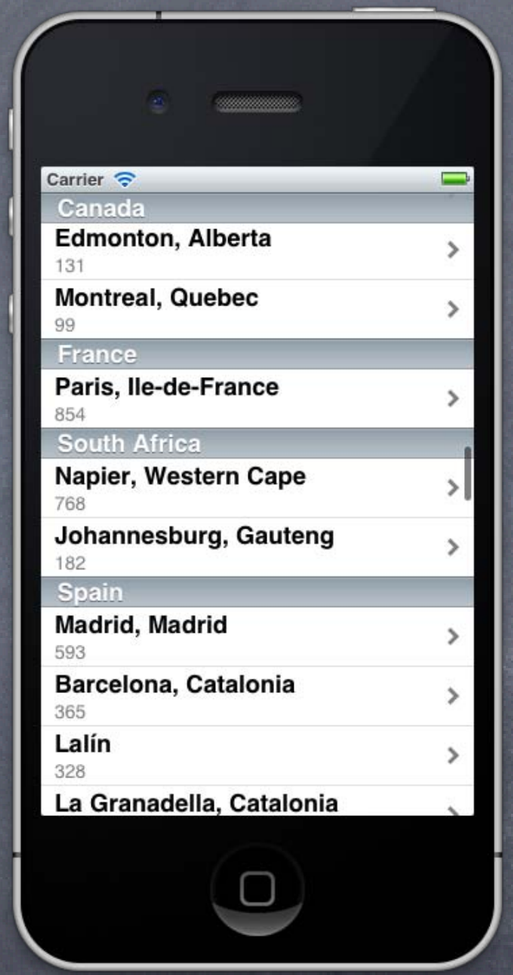
Table Cell

Section Footer

Sections or Not

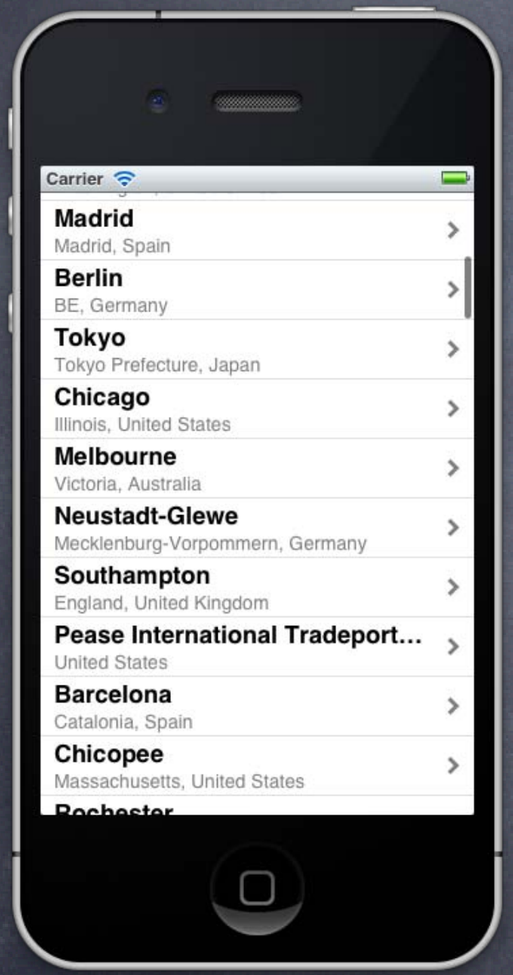


No Sections



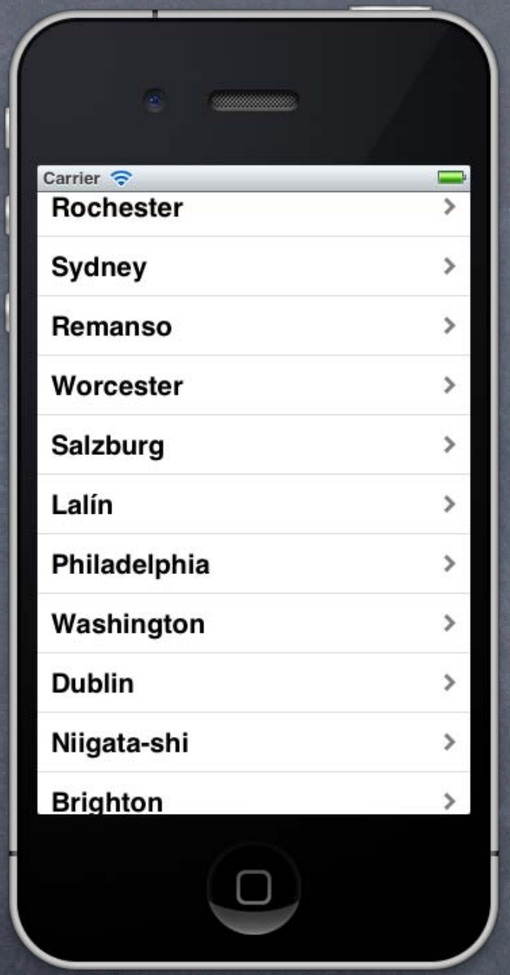
Sections

Cell Type



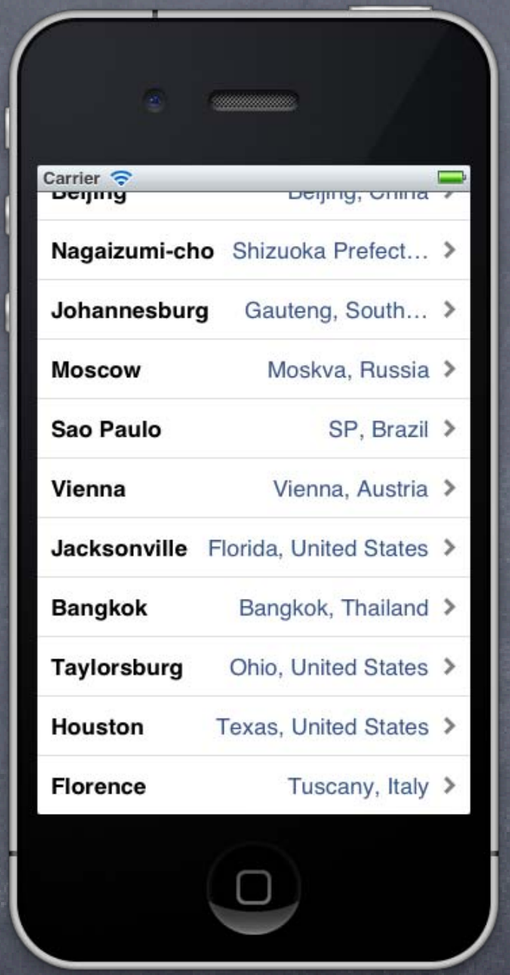
Subtitle

UITableViewCellStyleSubtitle



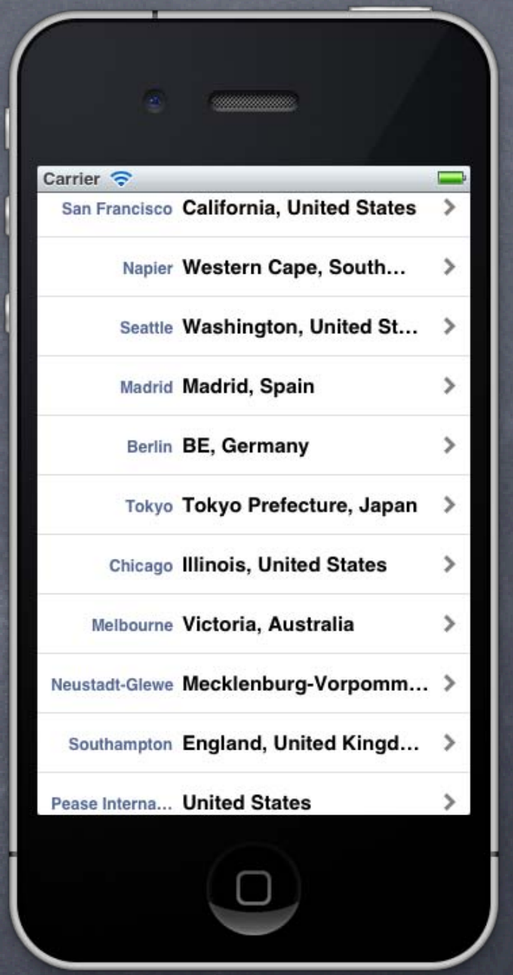
Basic

UITableViewCellStyleDefault



Right Detail

UITableViewCellStyleValue1



Left Detail

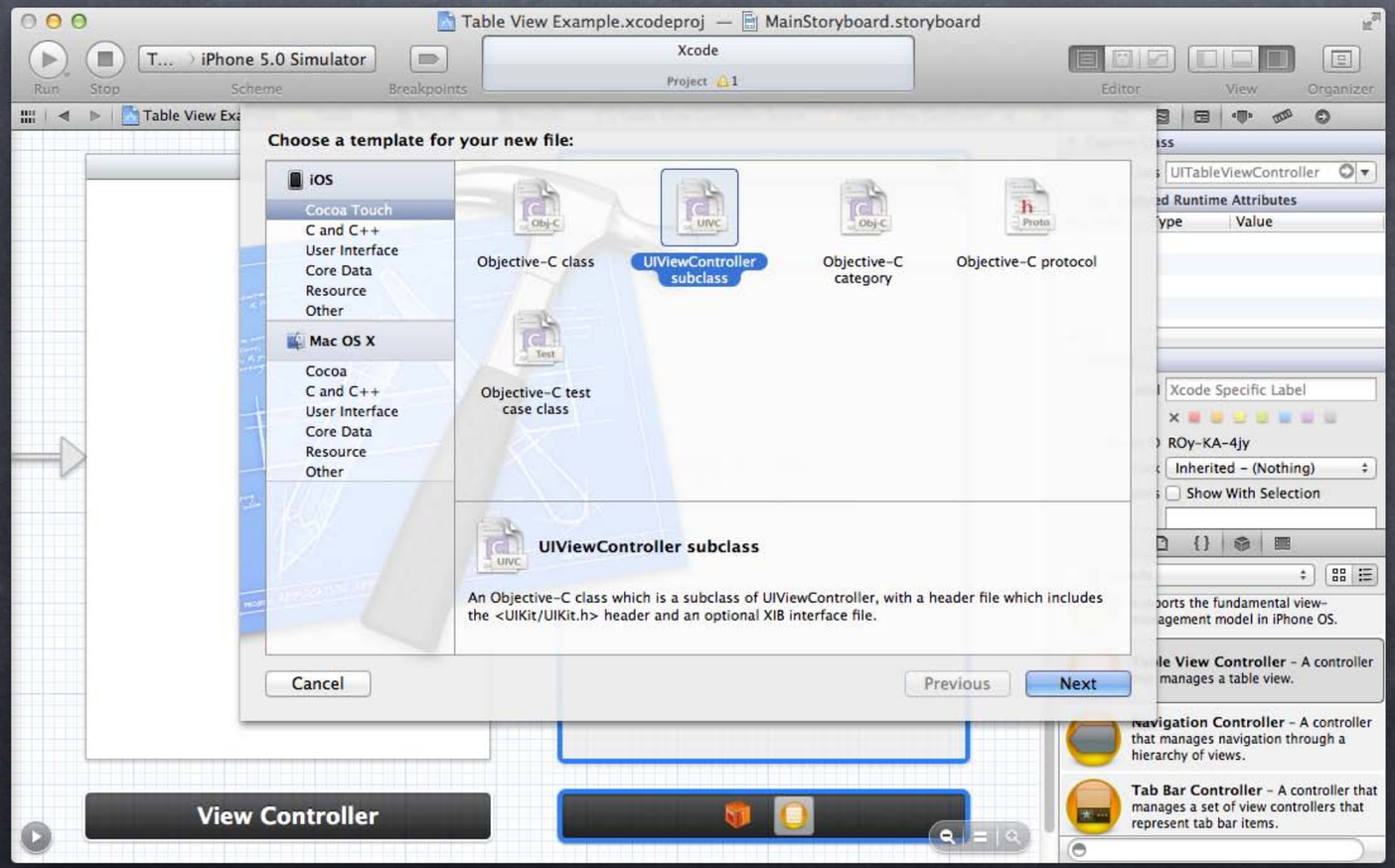
UITableViewCellStyleValue2

Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just ~~drag one out in Xcode,~~ create a subclass of it and you're on your way!



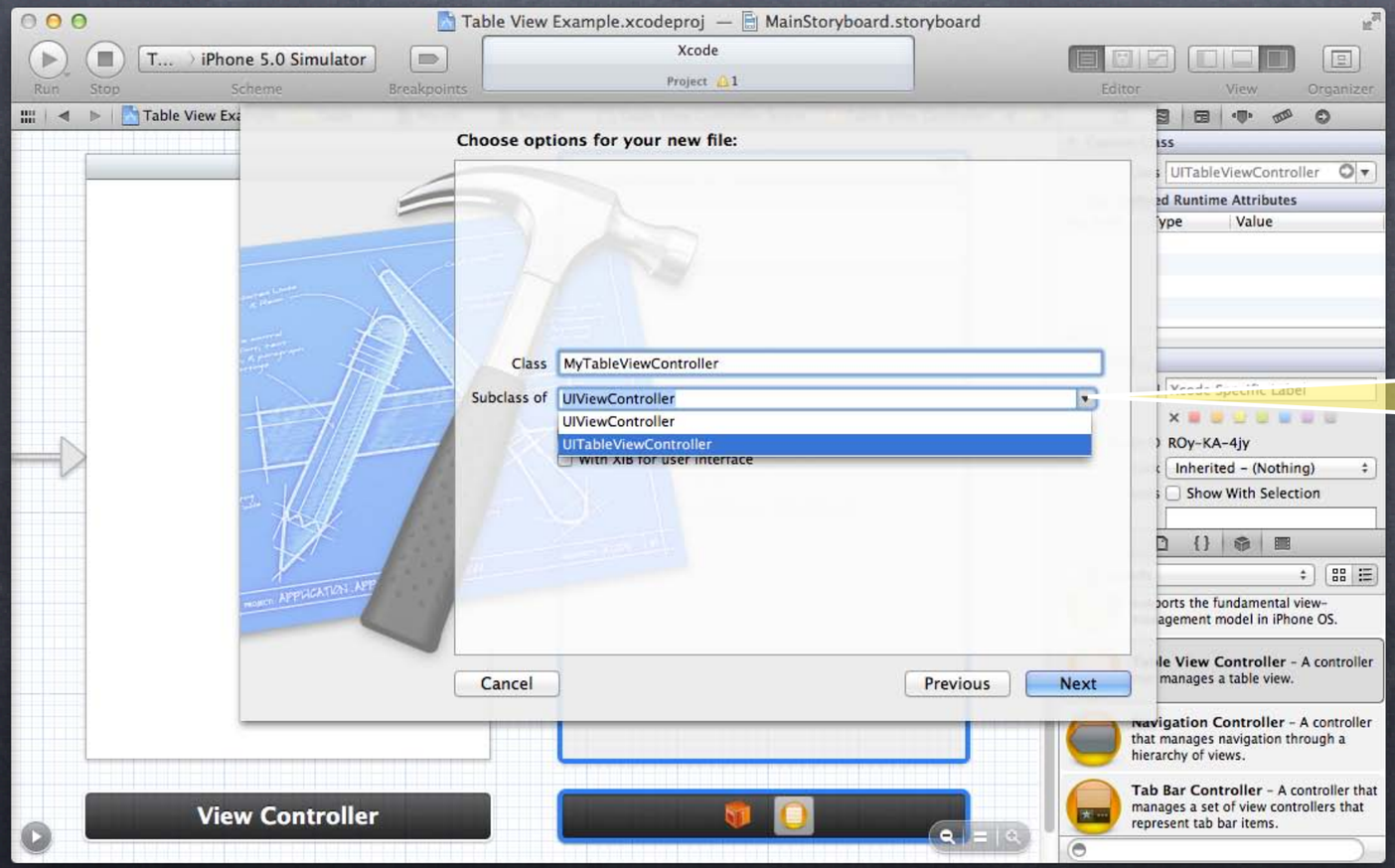
Choose New File ... from the File menu to create a custom subclass of `UITableViewController`

Creating Table View MVCs

- `UITableViewController`

  iOS class used as the base class for MVC's that display `UITableViews`

  Just ~~drag one out in Xcode,~~ create a subclass of it and you're on your way!



Choose New File … from the File menu to create a custom subclass of `UITableViewController`

Be sure to set the superclass to `UITableViewController`!!

UITableViewCell

- Cell Accessory
  - None
  - Disclosure Indicator
  - Checkmark
  - Detail Disclosure Indicator (= Button)


```
- (void)tableView:(UITableView *)tableView
accessoryTappedForRowAtIndexPath:(NSIndexPath *)indexPath
```

- ImageView

```
@property (nonatomic, strong) UIImageView *imageView
```

UITableView Protocols

- How do we connect to all this stuff in our code?

  A `UITableView` has two important `@property`s: its `delegate` and its `dataSource`.

  The `delegate` is used to control how the table is displayed.

  The `dataSource` provides the data what is displayed inside the cells.

  Your `UITableViewController` is automatically set as the `UITableView`'s `delegate` & `dataSource`.

  Your `UITableViewController` subclass will also have a property that points to the `UITableView`:

```
@property (nonatomic, strong) UITableView *tableView;
```

- To be "dynamic," we need to be the `UITableView`'s `dataSource`

  Three important methods in this protocol:

  How many sections in the table?

  How many rows in each section?

  Give me a `UIView` to use to draw each cell at a given row in a given section.

  Let's cover the last one first …

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?
  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).
  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

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


}
```

In a static table, you do not need to implement this method (though you can if you want to ignore what's in the storyboard).

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?
  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).
  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

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


}
```

NSIndexPath is just an object with two important properties for use with UITableView: row and section.

```
@interface NSIndexPath

@property (nonatomic) NSInteger section;
@property (nonatomic) NSInteger row;

+ (NSIndexPath *)indexPathForRow:(NSInteger)row inSection:(NSInteger)section;

@end
```

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    // set @propertys on the cell to prepare it to display
}
```

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"My Table View Cell"];

    // set @properties on the cell to prepare it to display
}
```

The cells in the table are actually reused.
When one goes off-screen, it gets put into a "reuse pool."
The next time a cell is needed, one is grabbed from the reuse pool if available.
If none is available, one will be put into the reuse pool if there's a prototype in the storyboard.
Otherwise this dequeue method will return `nil` (let's deal with that ...).

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
         cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@“My Table View Cell”];
    if (!cell) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
                                      reuseIdentifier:@“My Table View Cell”];
    }

    // set @propertys on the cell to prepare it to display
}
```

UITableViewDataSource

- How do we control what is drawn in each cell in a dynamic table?

  Each row is drawn by its own instance of `UITableViewCell` (a `UIView` subclass).

  Here is the `UITableViewDataSource` method to get that cell for a given row in a given section.

```
- (UITableViewCell *)tableView:(UITableView *)sender
        cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // get a cell to use (instance of UITableViewCell)
    UITableViewCell *cell;
    cell = [self.tableView dequeueReusableCellWithIdentifier:@"My Table View Cell"];
    if (!cell) {
        cell = [[UITableViewCell alloc] initWithStyle:UITableViewCellStyleSubtitle
                                      reuseIdentifier:@"My Table View Cell"];
    }
    cell.textLabel.text = [self getMyDataForRow:indexPath.row inSection:indexPath.section];
    return cell;
}
```

There are obviously other things you can do in the cell besides setting its text (detail text, image, checkmark, etc.).

UITableViewDataSource

- How does a dynamic table know how many rows there are?
  And how many sections, too, of course?
  ```
  - (NSInteger)numberOfSectionsInTableView:(UITableView *)sender;
  - (NSInteger)tableView:(UITableView *)sender numberOfRowsInSection:(NSInteger)section;
  ```
- Number of sections is 1 by default
  In other words, if you don't implement `numberOfSectionsInTableView:`, it will be 1.
- No default for number of rows in a section
  This is a required method in this protocol (as is `tableView:cellForRowAtIndexPath:`).
- What about a static table?
  Do not implement these `dataSource` methods for a static table.
  `UITableViewController` will take care of that for you.

UITableViewDataSource

- There are a number of other methods in this protocol
  But we're not going to cover them today.
  They are mostly about getting the headers and footers for sections.
  And about dealing with editing the table (moving/deleting/inserting rows).

UITableViewDelegate

- All of the above was the UITableView's dataSource
  But UITableView has another protocol-driven delegate called its delegate.
- The delegate controls how the UITableView is displayed
  Not what it displays (that's the dataSource's job).
- Common for dataSource and delegate to be the same object
  Usually the Controller of the MVC in which the UITableView is part of the (or is the entire) View.
- The delegate also lets you observe what the table view is doing
  Especially responding to when the user selects a row.
  We often will use segues when this happens, but we can also track it directly.

Table View "Target/Action"

- UITableViewDelegate method sent when row is selected

  This is sort of like "table view target/action"

  You might use this to update a detail view in a split view if master is a table view

```
- (void)tableView:(UITableView *)sender didSelectRowAtIndexPath:(NSIndexPath *)path
{
    // go do something based on information
    // about my data structure corresponding to indexPath.row in indexPath.section
}
```

- Lots and lots of other delegate methods

  will/did methods for both selecting and deselecting rows

  Providing UIView objects to draw section headers and footers

  Handling editing rows (moving them around with touch gestures)

  willBegin/didEnd notifications for editing.

  Copying/pasting rows.

UITableView

- What if your Model changes?

```
- (void)reloadData;
```

Causes the table view to call `numberOfSectionsInTableView:` and `numberOfRowsInSection:` all over again and then `cellForRowAtIndexPath:` on each visible cell.

Relatively heavyweight obviously, but if your entire data structure changes, that's what you need.

If only part of your Model changes, there are lighter-weight reloaders, for example …

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

- There are dozens of other methods in `UITableView`

Setting headers and footers for the entire table

Controlling the look (separator style and color, default row height, etc.)

Getting cell information (cell for index path, index path for cell, visible cells, etc.)

Scrolling to a row

Selection management (allows multiple selection, getting the selected row, etc.)

Moving, inserting and deleting rows, etc.

Further Literature

- UITableView Class Reference, Apple
  http://developer.apple.com/library/ios/#documentation/uikit/reference/UITableView_Class/Reference/Reference.html