Advanced Mobile Application Development
Week 3: Table Views

Table Views

https://developer.apple.com/ios/human-interface-guidelines/ui-views/tables/
The UITableViewController class manages the table view
https://developer.apple.com/reference/uikit/uitableviewcontroller
The UITableView class displays a table view https://developer.apple.com/reference/uikit/uitableview
A table presents data as a scrolling, single-column list of rows that can be divided into sections or groups.
Tables should be used to display large or small amounts of information the form of a list.
In conjunction with navigation controllers they are used to navigate through hierarchical data.
Table views have two styles: plain or grouped

- The plain style can also have sections with an index along the right side
- The grouped style must have at least 1 group and each group must have at least 1 item
- Both styles can have a header and footer

Table Rows

Each row in a table is a cell.
UITableViewCell class https://developer.apple.com/reference/uikit/uitableviewcell
There are four standard table cell styles or you can create a custom one.

1. Default: a simple cell style that has a single title left aligned and an optional image (UITableViewCellStyleDefault)
2. Subtitle : left-aligns the main title and puts a gray subtitle left aligned under it. It also permits an image in the default image location (UITableViewCellStyleSubtitle)
3. Value 1: left-aligns the main title and puts the subtitle in blue text and right-aligns it on the right side of the row. (UITableViewCellStyleValue1)
4. Value 2: puts the main title in blue and right-aligns it at a point that's indented from the left side of the row. The subtitle is left-aligned at a short distance to the right of this point. This style does not allow images. (UITableViewCellStyleValue2)

The **UITableViewDelegate** protocol handles configuring and managing table views.
The **UITableViewDataSource** protocol handles the row data in table views.
Table views can have an unlimited number of rows but only display only a few rows at a time.
In order to show data in the tables quickly, table views don't load all the rows, only the ones that need to be displayed at the time.
Then as rows scroll off the screen they are placed in a queue to be reused.
tableView.dequeueReusableCell(withIdentifier: for:) dequeues an existing cell if one is available or creates a new one

- The reuse identifier is a string used to identify a cell that is reusable

Reusing cells is the best way to guarantee smooth table view scrolling performance.

Search

https://developer.apple.com/ios/human-interface-guidelines/ui-bars/search-bars/
The UISearchController class incorporates a search bar, UISearchBar, into a view controller that has data

- In a table you can assign the search bar to the tableHeaderView property
- The searchResultsUpdater property is provided the search results from the search controller

The UISearchResultsUpdating protocol must be adopted and its 1 required method handles the search bar interaction

scrabbleQ
File | New Project
Single View App
iPhone

Go into MainStoryboard. The initial scene is a view controller but we want a table view controller.
Click on the scene and delete it. Then drag onto the canvas a table view controller.
In the attributes inspector check Is Initial View Controller.

We want our class to be the controller so go into ViewController.swift and change its super class to `UITableViewController.`
Now go back into MainStoryboard and select the view controller and change its class to ViewController.
Select the table view and note that it has the class UITableView.
In the Connections inspector check that the dataSource and delegate for the table view are set to View Controller. If not, drag from the circles to the View Controller icon.
In the attributes inspector see that the table view's style is plain. Try changing it to grouped and see how that looks.
Select a table view cell and in the attributes inspector you can see that the Table View Cell style is custom. We'll look at the others in a minute.
Select the Table View Cell and in the attributes inspector make the identifier "CellIdentifier".

If you run it at this point you should see a blank table view.

Add qwordswithoutu1.plist into your project and make sure Copy Items if Needed is checked as well as your project target.

Go into ViewController.swift and add an array that will hold the letters and words.
```
var words = [String]()
```

We'll create the same function as we did for the picker data to get the data loaded
```
    func getDataFile() -> String? {
        //use a Bundle object of the directory for our application to
retrieve the pathname of artistalbums.plist
        guard let pathString = Bundle.main.path(forResource:
"qwordswithoutu1", ofType: "plist") else {
            return nil
        }
        return pathString
    }
```

Then we load the data

```
    override func viewDidLoad() {
        super.viewDidLoad()
        guard let path = getDataFile() else{
            print("Error loading file")
            return
        }
        //load the words of the plist file into the array
        words = NSArray(contentsOfFile: path) as! Array
}
```

Now we implement the required methods for the UITableViewDataSource protocol

```
//Required methods for UITableViewDataSource
// Customize the number of rows in the section
    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return words.count
    }

// Displays table view cells
    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        //dequeues an existing cell if one is available, or creates a
new one and adds it to the table
        let cell = tableView.dequeueReusableCell(withIdentifier:
"CellIdentifier", for: indexPath)
        cell.textLabel?.text = words[indexPath.row]
        return cell
    }
```

You should now see your table view with the list of words.
To fix the fact that the table scrolls under the status bar
Add to viewDidLoad()

```
self.tableView.contentInset = UIEdgeInsetsMake(20, 0, 0, 0);
```

Looks good initially but it still scrolls under the status bar
Can't seem to set constraints on the table view.
But table views are usually in navigation controllers and that will fix the problem

Now let's add an image.
Drag scrabble_q_tile.png into Assets.xcassets
In MainStoryboard select the table view cell and change the style to basic and under image choose scrabble_q_tile.png.
You can also do the image programmatically

```
cell.imageView?.image=UIImage(named: "scrabbletile90.png")
```

Now when you run it you'll see the image.

Now change the Table View Cell style to subtitle.
Notice this adds a detail label.

Make its text say Q no U. (You can either add it in the label in the storyboard or do it programmatically)
`cell.detailTextLabel?.text="Q no U"`
Now when you run it you'll see a subtitle as well.

Now what if you want to do something when the user selects a row. It's a UITableViewDelegate method that handles this.

```
    //UITableViewDelegate method that is called when a row is selected
    override func tableView(_ tableView: UITableView, didSelectRowAt
indexPath: IndexPath) {
        let alert = UIAlertController(title: "Row selected", message:
"You selected \(words[indexPath.row])", preferredStyle:
UIAlertControllerStyle.alert)
        let okAction = UIAlertAction(title: "OK", style:
UIAlertActionStyle.default, handler: nil)
        alert.addAction(okAction)
        present(alert, animated: true, completion: nil)
        tableView.deselectRow(at: indexPath, animated: true)
//deselects the row that had been choosen
    }
```

search
Let's add the ability to search our table view by creating a new view controller class to handle search and its results.

File | New File
iOS | Cocoa Touch class
SearchResultsController
Subclass UITableViewController
Leave Also create xib file unchecked
Save it in your project and target

Go into your new file and adopt the UISearchResultsUpdating protocol
```
class SearchResultsController: UITableViewController,
UISearchResultsUpdating
```

(you'll get an error until we conform to the protocol)
SearchResultsController needs access to the list of words that the main view controller is displaying, so we'll need an array to store those words as well as an array to store the results of a search.

```
    var allwords = [String]()
    var filteredWords = [String]()
```

The file already contains some template code that provides a partial implementation of the UITableViewDataSource protocol and some commented-out methods UITableViewController subclasses often need. We're not going to use most of them so you can delete them if you want.

Since we won't have a scene for this view controller in our storyboard we need to register our cell reuse identifier programmatically. Add to viewDidLoad()

```
//register our table cell identifier
tableView.register(UITableViewCell.self, forCellReuseIdentifier:
"CellIdentifier")
```

The `UISearchResultsUpdating` protocol only has 1 method and it's required

```
    //UISearchResultsUpdating protocol required method to implement the
search
    func updateSearchResults(for searchController:
UISearchController) {
        let searchString = searchController.searchBar.text //search
string
        filteredWords.removeAll(keepingCapacity: true) //removes all
elements
        if searchString?.isEmpty == false {
            //closure that will be called for each word to see if it
matches the search string
            let filter: (String) -> Bool = { name in
                //look for the search string as a substring of the
word
                let range = name.range(of: searchString!, options:
NSString.CompareOptions.caseInsensitive, range: nil, locale: nil)
                return range != nil //returns true if the value
matches and false if there's no match
            } //end closure
            let matches = allwords.filter(filter)
            filteredWords.append(contentsOf: matches)
        }
        tableView.reloadData() //reload table data with search
results
    }
```

Then we need to implement the UITableViewDataSource methods to display the table view cells. We want the table rows to show the search results. Since SearchResultsController is a subclass of UITableViewController it automatically acts as the table's data source.

```
    override func tableView(_ tableView: UITableView,
numberOfRowsInSection section: Int) -> Int {
        return filteredWords.count
    }

    override func tableView(_ tableView: UITableView, cellForRowAt
indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier:
"CellIdentifier", for: indexPath)
        cell.textLabel?.text = filteredWords[indexPath.row]
        return cell
    }
```

Note that this method is included with a return of 0. You either need to change it to return 1 section or comment/delete it as 1 is the default if the method isn't present.

```
    override func numberOfSections(in tableView: UITableView) -> Int
{
    return 1
    }
```

Now we have to set up our main ViewController to add the search bar.
In ViewController.swift add an instance of UISearchController

```
    var searchController : UISearchController!
```

Update viewDidLoad() to implement and configure the search bar.

```
        //search results
        let resultsController = SearchResultsController() //create an
instance of our SearchResultsController class
        resultsController.allwords = words //set the allwords
property to our words array
        searchController =
UISearchController(searchResultsController: resultsController)
//initialize our search controller with the resultsController when it
has search results to display

        //search bar configuration
        searchController.searchBar.placeholder = "Enter a search
term" //place holder text
        searchController.searchBar.sizeToFit() //sets appropriate
size for the search bar
        tableView.tableHeaderView=searchController.searchBar
//install the search bar as the table header
        searchController.searchResultsUpdater = resultsController
//sets the instance to update search results
```

Each time the user types something into the search bar, UISearchController uses the object stored in its searchResultsUpdater property to update the search results.

Now you should be able to search through the data in your table view. Very useful for tables with a lot of data.

Grouped
Now let's look at the table view grouped style. (scrabbleQgrouped)
This uses qwordswithoutu2.plist which is a dictionary with keys that are letters and values that are an array of words. The keys are used to group the table and create an index.
In the storyboard the only difference is in the attributes inspector for the table view the style is Grouped.

In ViewController.swift there's a Dictionary for all the words and an Array for the letters

In viewDidLoad we load the plist into the Dictionary and then extract the keys into the letters array and sort them.

The Array class has a method called sort that sorts the array and assigns it back to itself. It takes a closure for how to do the sort. This is the shorthand way to sort Strings in ascending order. For more complex sorts you can write a function.

The delegate methods now all need to take into account the section.

`tableView(_:numberOfRowsInSection:)` has to calculate the number of words for a given section.

`tableView(_: cellForRowAt:)` and `tableView(_:, didSelectRowAt:)` have to get the section before the word.

`tableView(_: willDisplayHeaderView: forSection:)` called to display a section header

`numberOfSections (in:)` now has to be implemented to return the number of sections since it's no longer 1. (The default is 1 so we didn't need to implement it earlier)

`tableView(_: titleForHeaderInSection:)` sets the header value for each section.

`sectionIndexTitles(for:)` returns the letters to add as an index down the right side of the table.

`tableView(_: viewForHeaderInSection:)` an optional method that allows you to configure a custom view for the section headers.

We only have 4 different sections so it doesn't look great, but this is especially helpful for really long lists.