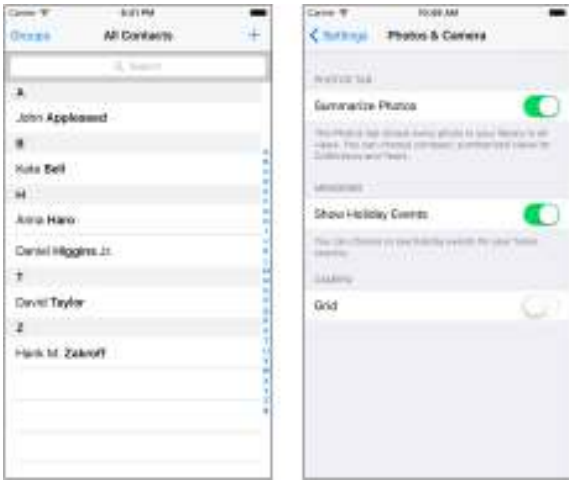
060010815:
iOS Application Development

Table View

Table View

- Tables display lists of data
- Each item in a table's list is a *row*
- Tables can have an *unlimited* number of rows
- Tables can be one column wide

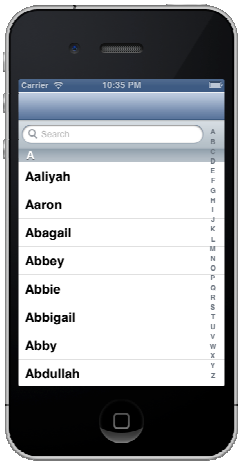
Examples of **UITableView**



Grouped Tables

- **Plain Tables.** The default style**.** May be *indexed*
- **Grouped Tables.** Each group is a set of rows embedded in a rounded rectangle.
  - **Can consist of a single group**
- Each division in a table is known to your data source as a **section**
  - In a grouped table, each group is a section
  - In an indexed table, each indexed grouping of data is a section. E.g., names beginning with ‘A’, names beginning with ‘B’, etc.

Indexed vs Grouped



Indexed Table

Grouped Table

Views vs Cells

- A *table view* is a subclass of a *view*
- Technically it is a *UITableView*
- Each visible row in a table is an instance of a *UITableViewCell*

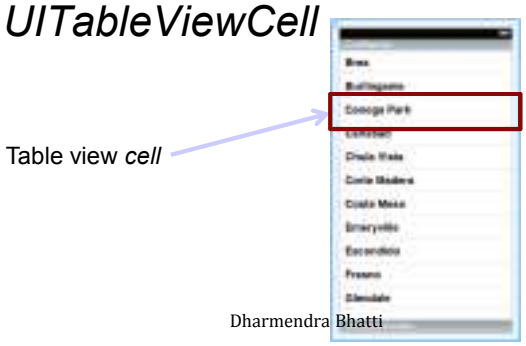


Table Data

- Table views do *not* store all the data
- Rather they just store data for the rows that are displayed

Table View – M or V or C ???

- Each class falls into exactly one of the following categories:
  - ***model***: holds data and knows nothing about the UI
  - ***view***: is visible to the user and knows nothing about the model objects
  - ***controller***: keeps the UI and the model objects in sync and controls the flow of the application

Table View – M or V or C ???

- As a view object, a **UITableView** does not handle application logic or data.

Table View – M or V or C ???

- A **UITableView** typically needs a view controller to handle its appearance on the screen.

Table View – M or V or C ???

- A **UITableView** needs a *data source*.
  - A **UITableView** asks its data source for the number of rows to display, the data to be shown in those rows, and other tidbits that make a **UITableView** a useful UI.
  - Without a data source, a table view is just an empty container.
  - The dataSource for a **UITableView** can be any type of object as long as it conforms to the **UITableViewDataSource** protocol.

Table View – M or V or C ???

- A **UITableView** typically needs a *delegate* that can inform other objects of events involving the **UITableView**. The delegate can be any object as long as it conforms to the **UITableViewDelegate** protocol.

Table Data

- They get their *configuration* data from the object that *conforms* to the *UITableViewdelegate* protocol
- They get their *row data* from the object that conforms to the *UITableViewDataSource* protocol

Delegates & Datasources

- **Delegate:** an object that takes responsibility for doing certain tasks on behalf of another object.
- A delegate contains methods that another object calls when certain events happen or when something needs to be done.

Delegates & Datasources

- **Datasource:** an object that supplies information to another object.
- A datasource knows about the model and can get information from it on behalf of view object

Delegates & Datasources

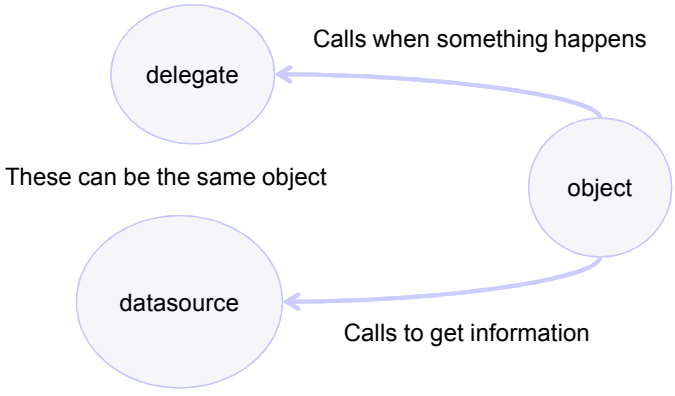


Table Cells

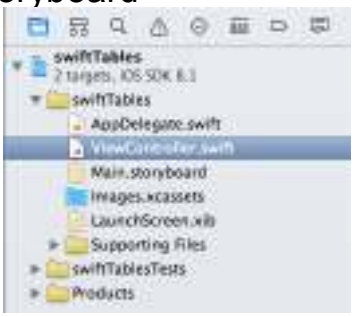
- Each row is represented by a *single UITableViewCell*
- Each *UITableViewCell* object can be configured with an image, some text, an optional accessory icon
- If you need more data (or columns) in a cell you can create a subview
  - Can do this programmatically
  - Can do this in IB

Creating DEMO

- Create a new project
  - Select *single view application*
  - Choose a name, etc. Make sure the language is *Swift* and it's for the *iPhone*
  - Leave unchecked: *use core data*
  - Choose a place to save
    - Do *not* need to use git

What did you get?

- Look at the file navigator
  - Normal appDelegate, viewcontroller, and Main.storyboard

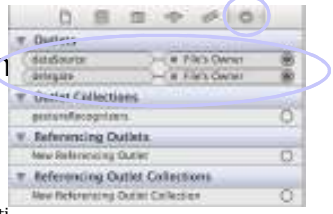


Creating the view

- Click on the Main.Storyboard to show IB
- Find a *Table View* (**not** a TableViewController) in the library and drag to the view.
- The *table view* should automatically size itself to fill the entire view.
- Click on the *pin* icon at bottom
  - Check the *Constrain to margins* checkbox
  - click on all 4 struts
  - change all 4 distances to 0
  - click the **Add 4 Constraints** button

Creating the view

- Open the hierarchical view by clicking on the Document Outline button
- Click on the Table View, bring up the *connections inspector*
  - Under the *outlets* section drag from *dataSource* to *View Controller* in the outline
  - Under the *outlets* section drag from *delegate* to *View Controller* in the outlin



This makes the viewController both the data source and delegate for the table.

Creating the controller

- in the Project Navigator click on the ViewController.swift file and add the bold:

This makes the view controlleraable to be both the data source and the delegate of a table.

```
class ViewController: UIViewController, UITableViewDataSource,
UITableViewDelegate {

 private let dwarves = [
            "Sleepy", "Sneezy", "Bashful", "Happy",
            "Doc", "Grumpy", "Dopey",
            "Thorin", "Dorin", "Nori", "Ori",
            "Balin", "Dwalin", "Fili", "Kili",
            "Oin", "Gloin", "Bifur", "Bofur",
            "Bombur"
    ]

    let simpleTableIdentifier = "SimpleTableIdentifier"
```

The array will hold our data. Normally the data will reside in the *model class* not the controller class.

UITableViewDataSource and **UITableViewDelegate** are *protocols*

Creating the controller II

- in the Project Navigator click on the ViewController.swift file and add the bold:

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

We need as many rows as we have array entries.

This method is called by the table when it needs to know how many rows are in the section (number of sections is 1 by default)

Creating the controller II

- in the Project Navigator click on the ViewController.swift file and add the bold:

This method is called by the table when it needs a cell to display. We must return a UITableViewCell

```
func tableView(tableView: UITableView,
        cellForRowAtIndexPath indexPath: NSIndexPath) ->

UITableViewCell {
```

Method body on next slide….

Creating the controller II

We check to see if there is an unused cell of the correct type that we can reuse.

```
var cell =
tableView.dequeueReusableCellWithIdentifier(simpleTableIdentifier)
as? UITableViewCell
if (cell == nil) {
        cell = UITableViewCell(style: UITableViewCellStyle.Default,
reuseIdentifier: simpleTableIdentifier)

cell!.textLabel?.text = dwarves[indexPath.row]
        }
```

The reuse identifier is used to identify the type of cell

If there was no leftover cell we have to create a new one.

Use default style; there are others!

The *indexPath* parameter contains the row (and section) number that the table will use this cell for.

The label text is what shows up in the row.

We return this cell; it will be displayed as the next row

As table rows scroll out of view, they are placed into a queue of cells to be reused. If memory gets low, they are discarded.

More on Cells

- Add image
  - Add icon image to Xcode
  - Change the method on next slide(assuming icon has name star.png)

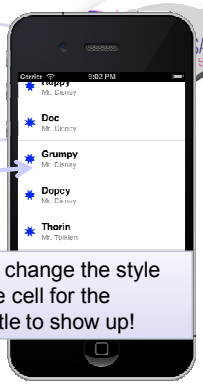
More on Cells

```
func tableView(tableView: UITableView,
      cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
         var cell =
tableView.dequeueReusableCellWithIdentifier(simpleTableIdentifier)
            as? UITableViewCell
         if (cell == nil) {
            cell = UITableViewCell(style: UITableViewCellStyle.Default,
reuseIdentifier: simpleTableIdentifier)
         }

         let image = UIImage(named: "star")
         cell!.imageView?.image = image
         let highlightedImage = UIImage(named: "star2")
         cell!.imageView?.highlightedImage = highlightedImage

         cell!.textLabel?.text = dwarves[indexPath.row]
         cell!.textLabel?.font = UIFont .boldSystemFontOfSize(50)
         return cell!
   }}
```


More on Cells

- Add detail text
  - Cells can have additional (“detail”) text.
  - Must change the cell “style”.

Subtitle



Must change the style of the cell for the subtitle to show up!

```
func tableView(tableView: UITableView,
      cellForRowAtIndexPath indexPath: NSIndexPath) -> UITableViewCell {
         var cell = tableView.dequeueReusableCellWithIdentifier(simpleTableIdentifier)
            as? UITableViewCell
         if (cell == nil) {
            cell = UITableViewCell(style: UITableViewCellStyle.Subtitle, reuseIdentifier: sim
         }

         let image = UIImage(named: "star")
         cell!.imageView?.image = image
         let highlightedImage = UIImage(named: "star2")
         cell!.imageView?.highlightedImage = highlightedImage

         if indexPath.row < 7 {
                  cell!.detailTextLabel?.text = "Mr Disney"
               } else {
                  cell!.detailTextLabel?.text = "Mr Tolkien"
               }

         cell!.textLabel?.text = dwarves[indexPath.row]
         cell!.textLabel?.font = UIFont .boldSystemFontOfSize(50)
         return cell!
   }}
```

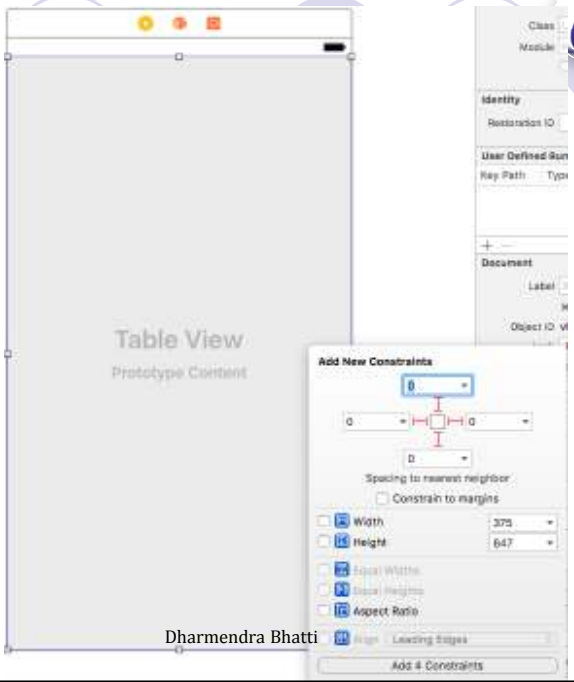
We’ll use different subtitles for the first 7 rows (0-6).

Other styles:
initWithStyle:UITableViewCellStyleValue1
initWithStyle:UITableViewCellStyleValue2

DEMO

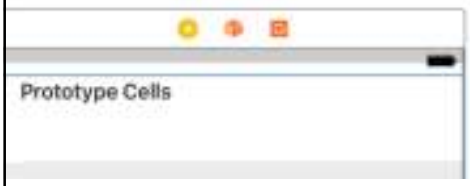
- Create a New Project “TableViewDemo”
- Select Devices “iPhone”
- Add “Table View” in main story board and set top, bottom, leading, trailing constraint to zero.

DEMO



DEMO

- Select “Table View” => attribute inspector and set Prototype Cells = 1

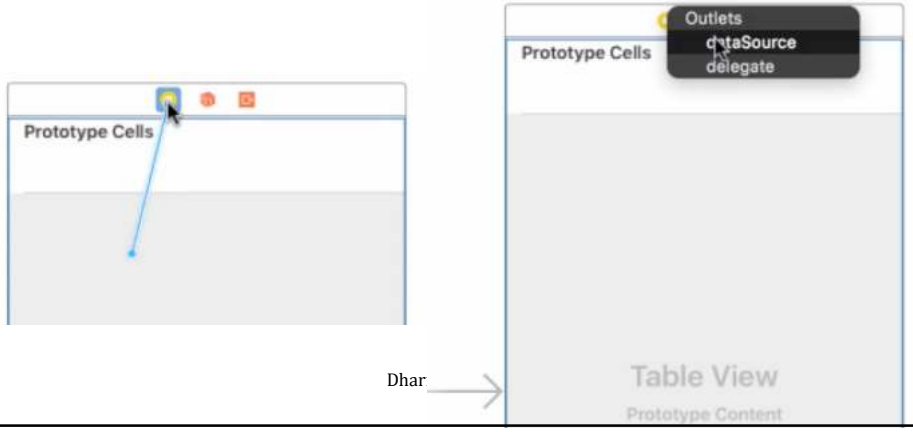


DEMO

- Select “Table View Cell” from Document Outline => attribute inspector => set identifier = “Cell”

DEMO

- Create two outlets dataSource and delegate by Ctrl+drag from Table View to View Controller



Dhar

DEMO

- ViewController.swift
  - Conform to the protocols
  - class ViewController: UIViewController, **UITableViewDelegate, UITableViewDataSource {**
  - **}**

DEMO

- UITableViewDataSource requires few methods
- Command + Click on UITableViewDataSource and copy first two methods in to your ViewController.swift
- Change both methods from public to internal

DEMO

- Create colorsArray in viewController class
- let colorsArray = ["Red", "Green", "Blue", "Yellow", "White", "Black", "Pink", "Purple"]

DEMO

- Create colorsArray in viewController class

```
override func viewDidLoad() {
        super.viewDidLoad()
        for index in 1...100
        {
                colorsArray.append("Color " + String(index))
        }
}
```


DEMO

- Add the code in first method

- internal func tableView(_ tableView: UITableView, **numberOfRowsInSection** section: Int) -> Int {

  return colorsArray.count

  }

DEMO

- Add the code in second method

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

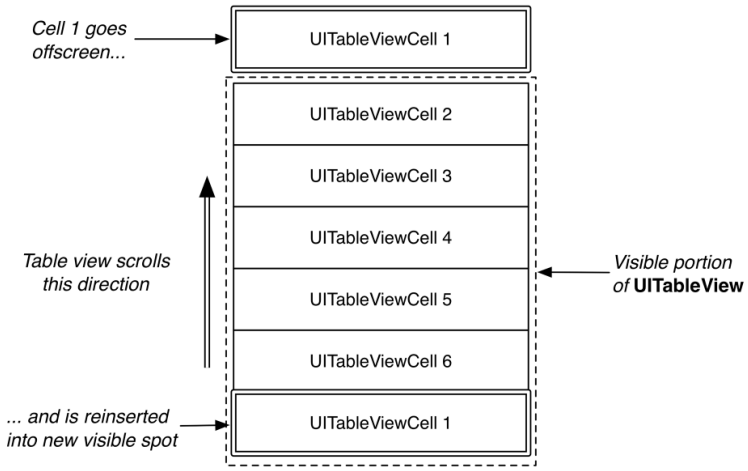
```
    let cell = UITableViewCell(style:
    UITableViewCellStyle.default, reuseIdentifier: "Cell")
    cell.textLabel?.text = colorsArray[indexPath.row]

    return cell
}
```


DEMO

- To reuse the cells already created:

- ~~let cell = UITableViewCell(style: UITableViewCellStyle.default, reuseIdentifier: "Cell")~~
- let cell = **tableView.dequeueReusableCell**(withIdentifier: "Cell", for: indexPath)

Reusable instances of **UITableViewCell**



Homepwner Application

Homepwner Application

- Create new Swift project “HomePwner”
- Create new Swift file “ItemsViewController.swift”

```
~~import Foundation~~
import UIKit

class ItemsViewController: UITableViewController {

}
```


Homepwner Application

- Now open Main.storyboard.
- Select the existing View Controller on the canvas and press Delete.
- Select the “ViewController.swift” and press Delete.

Homepwner Application

- Then drag a Table View Controller from the object library onto the canvas.
- With the Table View Controller selected, open its identity inspector and change the class to ItemsViewController.

Homepwner Application

- Finally, open the attributes inspector for Items View Controller and check the box for “Is Initial View Controller”.
- Build and run your application.

Homepwner Application

Homepwner Application

- Create a new Swift file named Item. In Item.swift, define the **Item** class and give it four properties.

```
~~import Foundation~~
import UIKit

class Item: NSObject {
    var name: String
    var valueInDollars: Int
    var serialNumber: String?
    let dateCreated: Date
}
```

Homepwner Application

- Classes can have two kinds of initializers: *designated initializers* and *convenience initializers*.
- Implement a new designated initializer on the **Item** class that sets the initial values for all of the properties.

Homepwner Application

```
import UIKit

class Item: NSObject {
    var name: String
    var valueInDollars: Int
    var serialNumber: String?
    let dateCreated: Date

    init(name: String, serialNumber: String?, valueInDollars: Int) {
        self.name = name
        self.valueInDollars = valueInDollars
        self.serialNumber = serialNumber
        self.dateCreated = Date()

        super.init()
    }
}
```

Homepwner Application

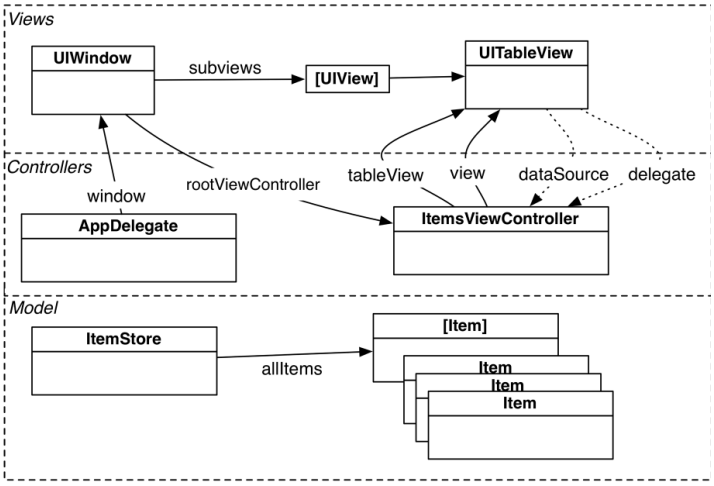
- Every class must have at least one designated initializer, but convenience initializers are optional.
- A convenience initializer always calls another initializer on the same class.
- Convenience initializers are indicated by the convenience keyword before the initializer name.

Homepwner Application

- Add a convenience initializer to **Item** that creates a randomly generated item.

```
convenience init(random: Bool = false) {
    if random {
        let adjectives = ["Fluffy", "Rusty", "Shiny"]
        let nouns = ["Bear", "Spork", "Mac"]

        var idx = arc4random_uniform(UInt32(adjectives.count))
        let randomAdjective = adjectives[Int(idx)]

        idx = arc4random_uniform(UInt32(nouns.count))
        let randomNoun = nouns[Int(idx)]

        let randomName = "\(randomAdjective) \(randomNoun)"
        let randomValue = Int(arc4random_uniform(100))
        let randomSerialNumber =
            UUID().uuidString.components(separatedBy: "-").first!

        self.init(name: randomName,
            serialNumber: randomSerialNumber,
            valueInDollars: randomValue)
    } else {
        self.init(name: "", serialNumber: nil, valueInDollars: 0)
    }
}
```

Homepwner Application



Homepwner Application

- Create a new Swift file named ItemStore. In ItemStore.swift, define the **ItemStore** class and declare a property to store the list of **Item**s.

```
~~import Foundation~~
import UIKit

class ItemStore {

    var allItems = [Item]()

}
```

Homepwner Application

- In ItemStore.swift, implement **createItem()** to create and return a new **Item**.

```
@discardableResult func createItem() -> Item {
    let newItem = Item(random: true)

    allItems.append(newItem)

    return newItem
}
```


Homepwner Application

- Give the controller access to the store
- In ItemsViewController.swift, add a property for an **ItemStore**.

```
class ItemsViewController: UITableViewController {

    var itemStore: ItemStore!
}
```

- set this property on the **ItemsViewController** when the application first launches

Homepwner Application

- Open AppDelegate.swift

```
func application(_ application: UIApplication, didFinishLaunchingWithOptions
        launchOptions: [UIApplicationLaunchOptionsKey : Any]?) -> Bool {
    // Override point for customization after application launch.

    // Create an ItemStore
    let itemStore = ItemStore()

    // Access the ItemsViewController and set its item store
    let itemsController = window!.rootViewController as! ItemsViewController
    itemsController.itemStore = itemStore

    return true
}
```


Homepwner Application

- In ItemStore.swift, implement the designated initializer to add five random items.

```
init() {
    for _ in 0..<5 {
        createItem()
    }
}
```

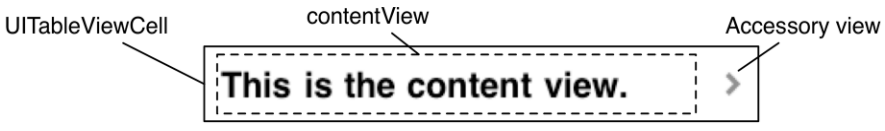
Homepwner Application

- In ItemsViewController.swift, implement **tableView(_:numberOfRowsInSection:)**.

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

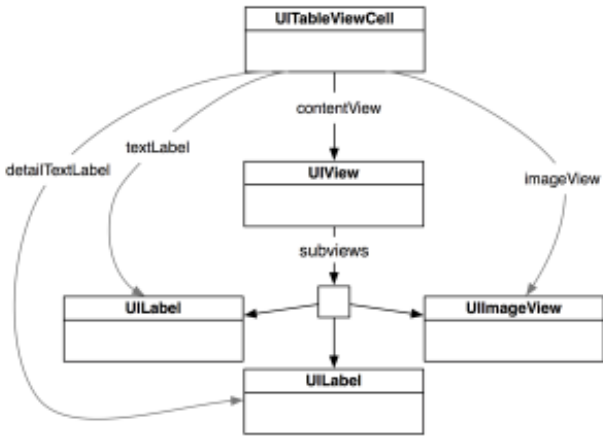

Homepwner Application

- Each row of a table view is a view.
- These views are instances of **UITableViewCell**.



Homepwner Application

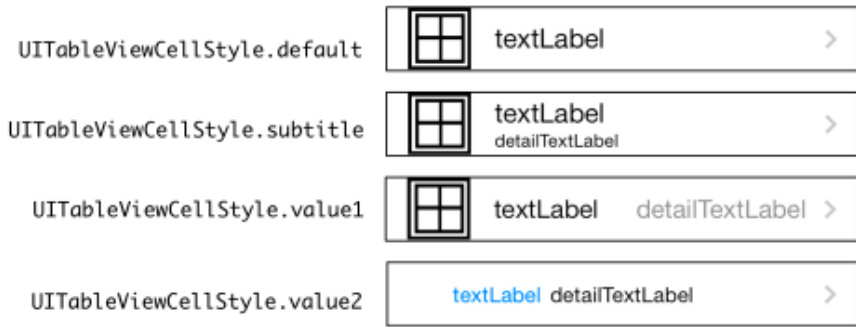
- **UITableViewCell** hierarchy



61

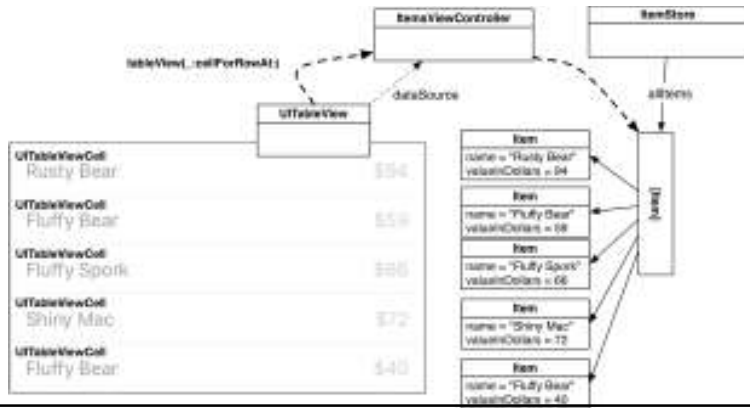
Homepwner Application

- **UITableViewCellStyle**: styles and constants



Homepwner Application

- Each cell will display the name of an **Item** as its textLabel and the valueInDollars of the **Item** as its detailTextLabel.



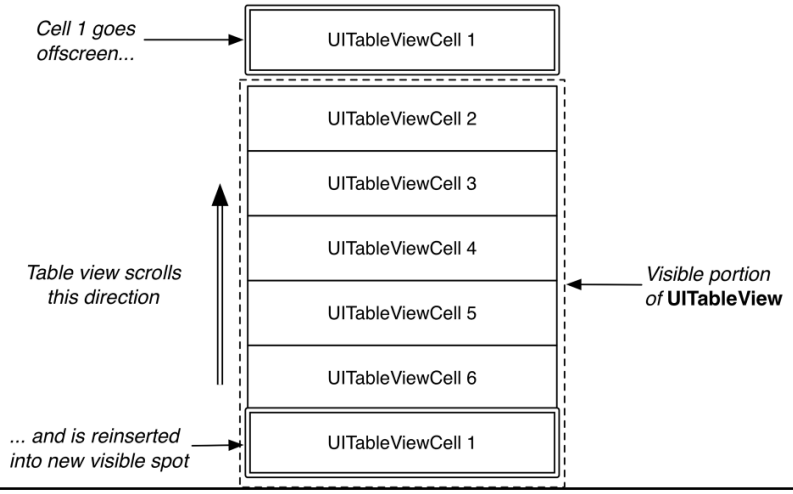
Homepwner Application

- In ItemsViewController.swift, implement **tableView(_:cellForRowAt:)**

```
override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    // Create an instance of UITableViewCell, with default appearance
    let cell = UITableViewCell(style: .value1, reuseIdentifier: "UITableViewCell")

    // Set the text on the cell with the description of the item
    // that is at the nth index of items, where n = row this cell
    // will appear in on the tableview
    let item = itemStore.allItems[indexPath.row]

    cell.textLabel?.text = item.name
    cell.detailTextLabel?.text = "$\(item.valueInDollars)"

    return cell
}
```

Homepwner Application

- Reusableinstancesof**UITableViewCell**



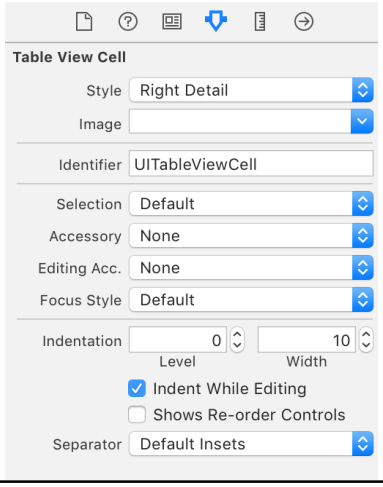
65

Homepwner Application

- Open Main.storyboard, Select “Prototype Cells”, and open its attributes inspector.
- Change the Style to “**Right Detail”** (which corresponds to **UITableViewCellStyle.value1**) and give it an Identifier of “**UITableViewCell”**

Homepwner Application

- Table view cell attributes



Homepwner Application

- In ItemsViewController.swift, update **tableView(_:cellForRowAt:)** to reuse cells.

```
override func tableView(_ tableView: UITableView,
        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    // Create an instance of UITableViewCell, with default appearance
    let cell = UITableViewCell(style: .value1, reuseIdentifier: "UITableViewCell")

    // Get a new or recycled cell
    let cell = tableView.dequeueReusableCell(withIdentifier: "UITableViewCell",
        for: indexPath)
    ...
}
```

Homepwner Application

- table view cells should not underlap the status bar
- Solution: **Content Insets**

Homepwner Application

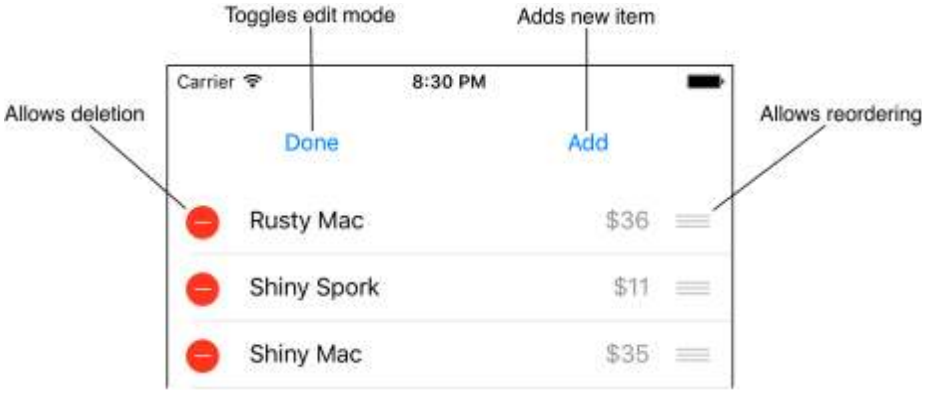
- In ItemsViewController.swift, override **viewDidLoad()**

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

    // Get the height of the status bar
    let statusBarHeight = UIApplication.shared.statusBarFrame.height

    let insets = UIEdgeInsets(top: statusBarHeight, left: 0, bottom: 0, right: 0)
    tableView.contentInset = insets
    tableView.scrollIndicatorInsets = insets
}
```

Editing UITableView

Homepwner Application

- Homepwner in editing mode



Editing Mode

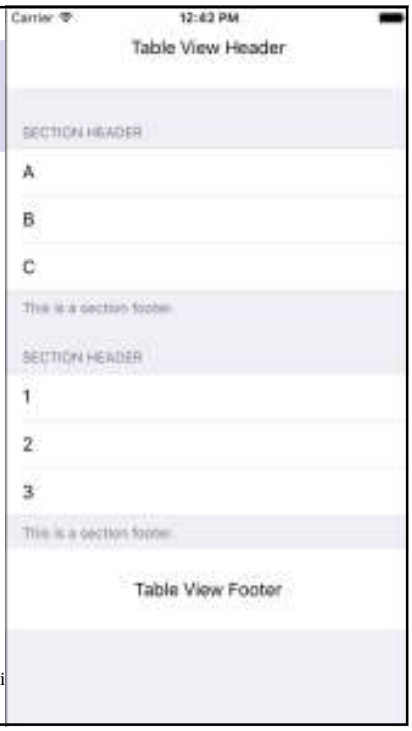
- **UITableView** has an editing property, and when this property is set to true, the **UITableView** enters editing mode.
- Once the table view is in editing mode, the rows of the table can be manipulated by the user.

Editing Mode

- Depending on how the table view is configured, the user can change the order of the rows, add rows, or remove rows.
- Editing mode does not allow the user to edit the *content* of a row.

Editing Mode

- Headers and footers
- Add a button in the *header view* of the table to put the **UITableView** in editing mode



Editing Mode

- In ItemsViewController.swift, stub out two methods in the implementation.

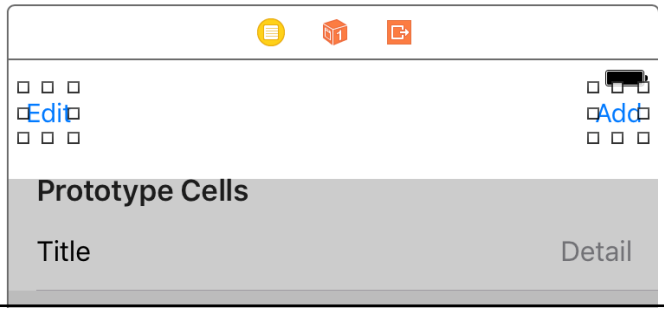
```
class ItemsViewController: UITableViewController {

    var itemStore: ItemStore!

    @IBAction func addNewItem(_ sender: UIButton) {

    }

    @IBAction func toggleEditingMode(_ sender: UIButton) {

    }
```

Editing Mode

- Open Main.storyboard.
- From the object library, drag a View to the very top of the table view, above the prototype cell.
- Resize the height of this view to be about 60 points.

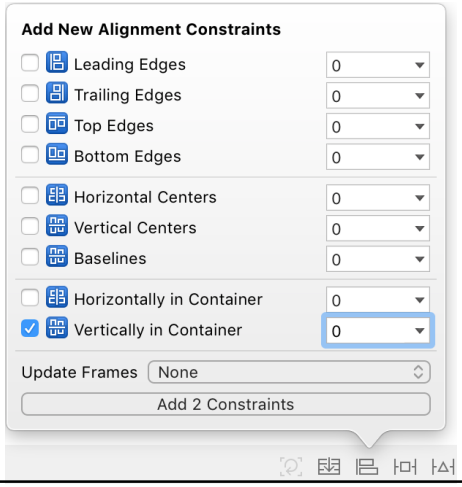
Editing Mode

- Drag two Buttons from the object library to the header view.
- Change their text and position them as shown in Figure



Align menu constraints

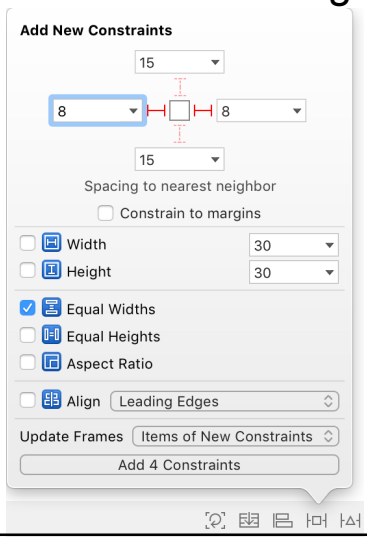
- Select both of the buttons and open the Auto Layout Align menu. Select Vertically in Container with a constant of 0.



79

Adding new constraints

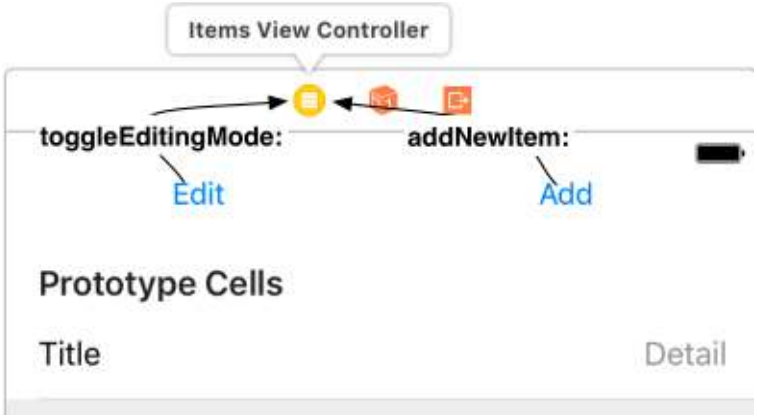
- Open the Add New Constraints menu and configure it as shown in Figure



80

Connecting the two actions

- connect the actions for the two buttons as shown in Figure



81

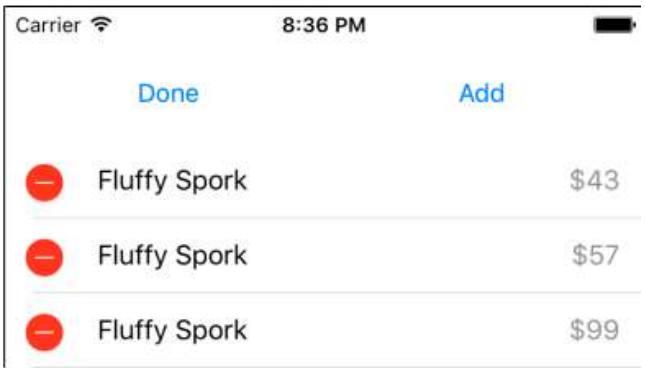
Homepwner Application

- In ItemsViewController.swift, implement **toggleEditingMode(_:)**.

```
@IBAction func toggleEditingMode(_ sender: UIButton) {
    // If you are currently in editing mode...
    if isEditing {
        // Change text of button to inform user of state
        sender.setTitle("Edit", for: .normal)

        // Turn off editing mode
        setEditing(false, animated: true)
    } else {
        // Change text of button to inform user of state
        sender.setTitle("Done", for: .normal)

        // Enter editing mode
        setEditing(true, animated: true)
    }
}
```

Homepwner Application

- **UITableView** in editing mode
- Editing Mode



Adding Rows

- When “Add” button is tapped, a new row will be added to the **UITableView**.
- In ItemsViewController.swift, implement **addNewItem(_:)**.

```
@IBAction func addNewItem(_ sender: UIButton) {
    // Create a new item and add it to the store
    let newItem = itemStore.createItem()

    // Figure out where that item is in the array
    if let index = itemStore.allItems.index(of: newItem) {
        let indexPath = IndexPath(row: index, section: 0)

        // Insert this new row into the table
        tableView.insertRows(at: [indexPath], with: .automatic)
    }
}
```

Homepwner Application

- Open ItemStore.swift and remove the initializer code.

```
init() {
    for _ in 0..<5 {
        createItem()
    }
}
```


Deleting Rows

- remove the row from the **UITableView** and
- remove the **Item** associated with it from the **ItemStore**
- In ItemStore.swift, implement a new method to remove a specific item.

```
func removeItem(_ item: Item) {
    if let index = allItems.index(of: item) {
        allItems.remove(at: index)
    }
}
```

Deleting Rows

- In ItemsViewController.swift

```
override func tableView(_ tableView: UITableView,
                        commit editingStyle: UITableViewCellEditingStyle,
                        forRowAt indexPath: IndexPath) {
    // If the table view is asking to commit a delete command...
    if editingStyle == .delete {
        let item = itemStore.allItems[indexPath.row]
        // Remove the item from the store
        itemStore.removeItem(item)

        // Also remove that row from the table view with an animation
        tableView.deleteRows(at: [indexPath], with: .automatic)
    }
}
```


Moving Rows

- To change the order of rows in a **UITableView**, use another method from the **UITableViewDataSource** protocol – **tableView(_:moveRowAt:to:)**.
- Give the **ItemStore** a method to change the order of items in its allItems array.

Moving Rows

- In ItemStore.swift, implement this new method.

```
func moveItem(from fromIndex: Int, to toIndex: Int) {
    if fromIndex == toIndex {
        return
    }

    // Get reference to object being moved so you can reinsert it
    let movedItem = allItems[fromIndex]

    // Remove item from array
    allItems.remove(at: fromIndex)

    // Insert item in array at new location
    allItems.insert(movedItem, at: toIndex)
}
```


Moving Rows

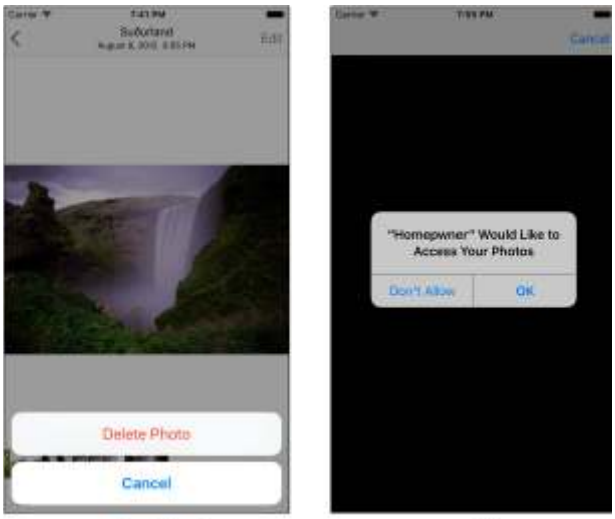
- In ItemsViewController.swift, implement **tableView(_:moveRowAt:to:)** to update the store.

```
override func tableView(_ tableView: UITableView,
                        moveRowAt sourceIndexPath: IndexPath,
                        to destinationIndexPath: IndexPath) {
    // Update the model
    itemStore.moveItem(from: sourceIndexPath.row, to: destinationIndexPath.row)
}
```

Displaying User Alerts

- Create an instance of **UIAlertController** to display an alert
  - **UIAlertControllerStyle.actionSheet**
  - **UIAlertControllerStyle.alert**

Displaying User Alerts



.actionSheet .alert

Displaying User Alerts

- In ItemsViewController.swift

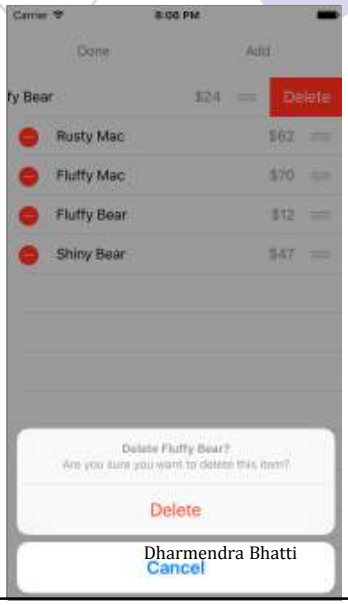
```
override func tableView(_ tableView: UITableView,
                        commit editingStyle: UITableViewCellEditingStyle,
                        forRowAt indexPath: IndexPath) {
    // If the table view is asking to commit a delete command...
    if editingStyle == .delete {
        let item = itemStore.allItems[indexPath.row]

        let title = "Delete \(item.name)?"
        let message = "Are you sure you want to delete this item?"

        let ac = UIAlertController(title: title,
                                   message: message,
                                   preferredStyle: .actionSheet)

        // Remove the item from the store
        itemStore.removeItem(item)

        // Also remove that row from the table view with an animation
        tableView.deleteRows(at: [indexPath], with: .automatic)
    }
}
```


Displaying User Alerts

- In **tableView(_:commit:forRowAt:)**

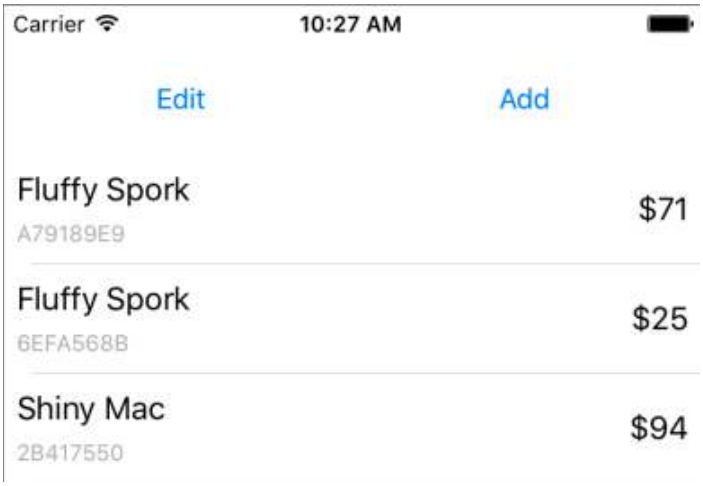
```
let ac = UIAlertController(title: title,
                           message: message,
                           preferredStyle: .actionSheet)

let cancelAction = UIAlertAction(title: "Cancel", style: .cancel, handler: nil)
ac.addAction(cancelAction)

let deleteAction = UIAlertAction(title: "Delete", style: .destructive,
                         handler: { (action) -> Void in
    // Remove the item from the store
    self.itemStore.removeItem(item)

    // Also remove that row from the table view with an animation
    self.tableView.deleteRows(at: [indexPath], with: .automatic)
})
ac.addAction(deleteAction)
// Present the alert controller
present(ac, animated: true, completion: nil)
...
```

Deleting an item



Subclassing UITableViewCell

Homepwner with subclassed table view cells

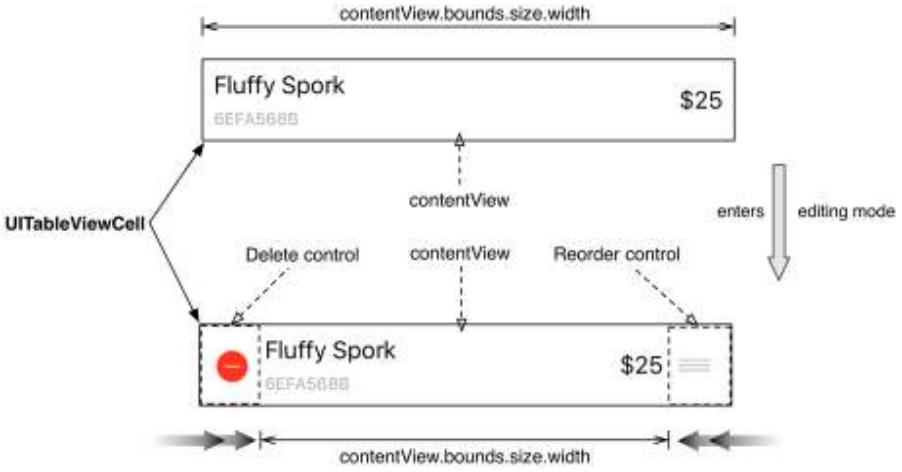


TableViewCell

- Customize the appearance of **UITableViewCell** subclasses by adding subviews to its contentView.
- When a table view enters editing mode, the contentView resizes itself to make room for the editing controls

TableViewCell

- Table view cell layout in standard and editing mode



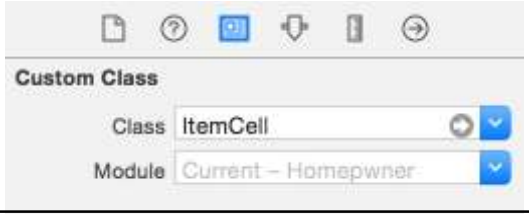
Creating ItemCell

- Create a new Swift file named ItemCell.

- ~~**import Foundation**~~
- **import UIKit**
- **class ItemCell: UITableViewCell {**
- **}**

Creating ItemCell

- Main.storyboard => UITableViewCell => attributes inspector => change the Style to Custom, and change the Identifier to ItemCell
- open its identity inspector => In the Class field, enter **ItemCell**



101

TableViewCell

- Change the height of the prototype cell to be about 65 points tall
- Drag three **UILabel** objects onto the cell.
- Configure them as shown in Figure.
- Make the text of the bottom label a slightly smaller font in a light shade of gray.

Add constraints to these three labels

- top-left label => Auto Layout => Add New Constraints => set top and left strut Constraints.
- Control-drag from the bottom-left label to the top-left label and select Leading.
- bottom-left label => Add New Constraints => set bottom strut Constraint.

Add constraints to these three labels

- Select the right label and Control-drag from this label to its superview on its right side.
- Select both Trailing Space to Container Margin and Center Vertically in Container.

Add constraints to these three labels

- Select the bottom-left label and open its size inspector.
- Find the Vertical Content Hugging Priority and lower it to 250.
- Lower the Vertical Content Compression Resistance Priority to 749.
- Select the three labels and click the Update Frames button.

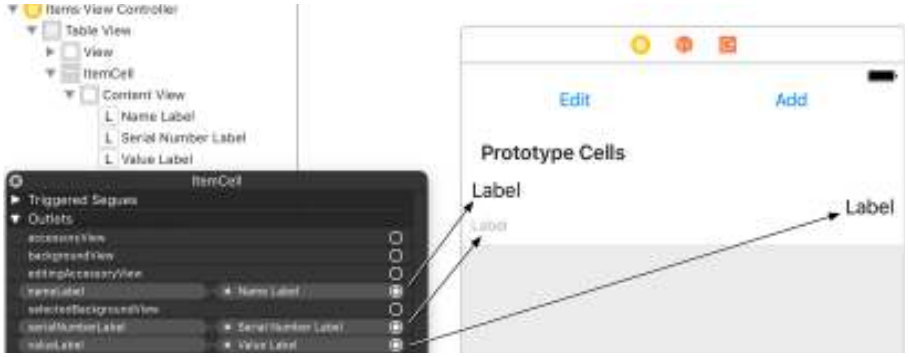
Exposing the Properties of ItemCell

- Open ItemCell.swift and add three properties for the outlets.

```
import UIKit

class ItemCell: UITableViewCell {

    @IBOutlet var nameLabel: UILabel!
    @IBOutlet var serialNumberLabel: UILabel!
    @IBOutlet var valueLabel: UILabel!

}
```

connect the outlets for the three views to the **ItemCell**

- Main.storyboard => Control-click on the ItemCell in the document outline and make the three outlet connections



ItemsViewController.swift

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

    // Get the height of the status bar
    let statusBarHeight = UIApplication.shared.statusBarFrame.height

    let insets = UIEdgeInsets(top: statusBarHeight, left: 0, bottom: 0, right: 0)
    tableView.contentInset = insets
    tableView.scrollIndicatorInsets = insets

    tableView.rowHeight = 65
}
```

ItemsViewController.swift, modify **tableView(_:cellForRowAt:)**

```
let cell = tableView.dequeueReusableCell(withIdentifier: "UITableViewCell",
    for: indexPath)

let cell = tableView.dequeueReusableCell(withIdentifier: "ItemCell",
    for: indexPath) as! ItemCell
cell.textLabel?.text = item.name
cell.detailTextLabel?.text = "$\(item.valueInDollars)"

// Configure the cell with the Item
cell.nameLabel.text = item.name
cell.serialNumberLabel.text = item.serialNumber
cell.valueLabel.text = "$\(item.valueInDollars)"
```

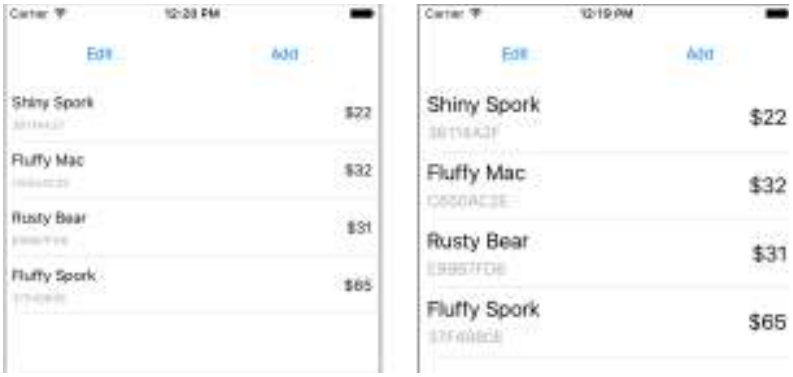

Dynamic Cell Heights

- Main.storyboard => Control-drag from the nameLabel to the serialNumberLabel => select Vertical Spacing
- Open ItemsViewController.swift and update **viewDidLoad()**

```
tableView.rowHeight = 65
tableView.rowHeight = UITableViewAutomaticDimension
tableView.estimatedRowHeight = 65
```

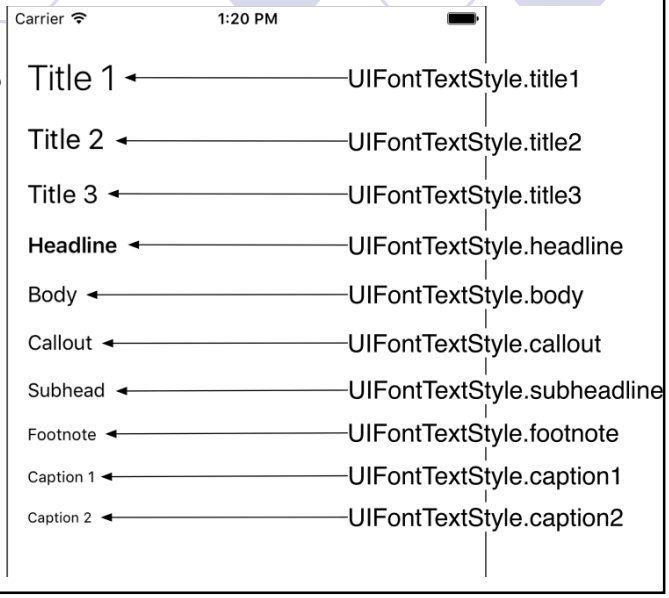
Dynamic Type

- ItemCell with smallest and largest user-selectable Dynamic Type sizes



Dynamic Type

- Text styles

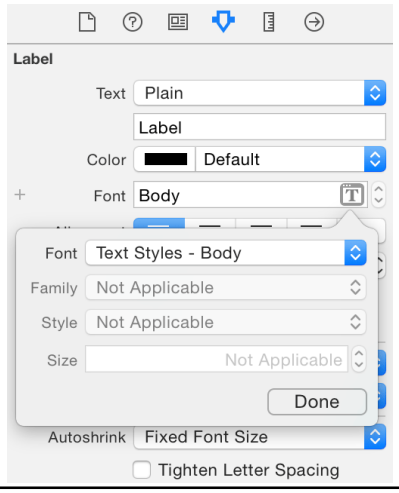


Dynamic Type

- Open Main.storyboard.
- Update the labels to **use the text styles** instead of **fixed fonts**.
- Select the nameLabel and valueLabel and open the attributes inspector.
- Click on the text icon to the right of Font.
- For Font, choose Text Styles - Body.
- Repeat the same steps for the serialNumberLabel, choosing the Caption 1 text style.

Dynamic Type

- Changing the text style



Dynamic Type

- Inform labels about change in font size
- Open ItemCell.swift and override **awakeFromNib()**

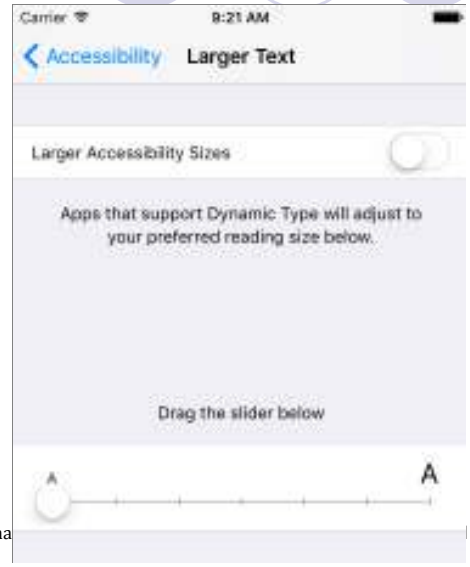
```
override func awakeFromNib() {
    super.awakeFromNib()

    nameLabel.adjustsFontForContentSizeCategory = true
    serialNumberLabel.adjustsFontForContentSizeCategory = true
    valueLabel.adjustsFontForContentSizeCategory = true
}
```


Build and run the application

- From the simulator's Hardware menu, select Home.
- On the simulator's Home screen, open the Settings application.
- Choose General, then Accessibility, and then Larger Text.
- Drag the slider all the way to the left to set the font size to the smallest value

Build and run the application

- Text size settings



Questions ???