CS275
Fall 2020

iOS: Subclassing `UITableViewCell`

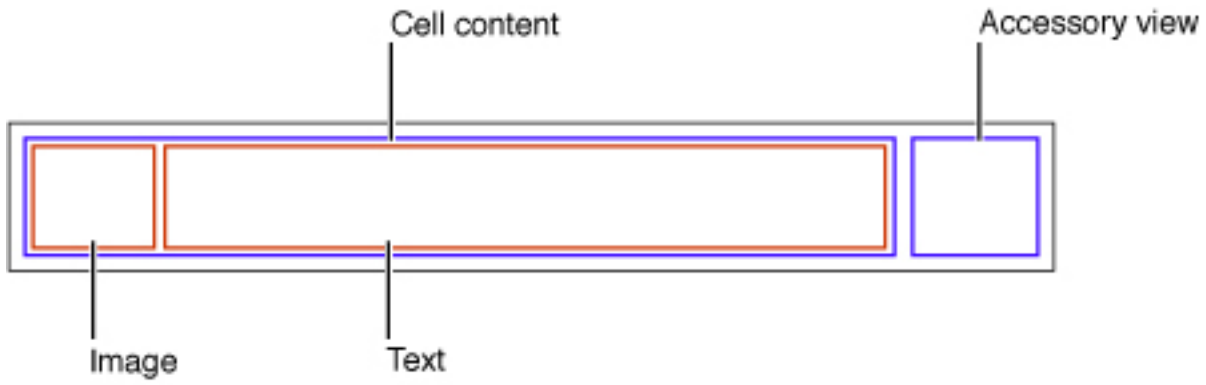
BNRG CHAPTER 12

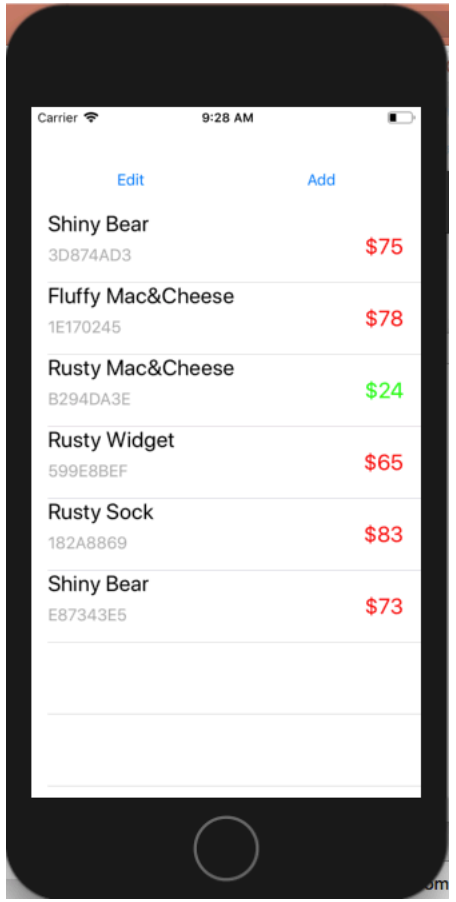
Topics

- Custom `UITableView` cells
- Dynamic Type

UITableViewCell

- The basic `UITableViewCell` has a predefined view

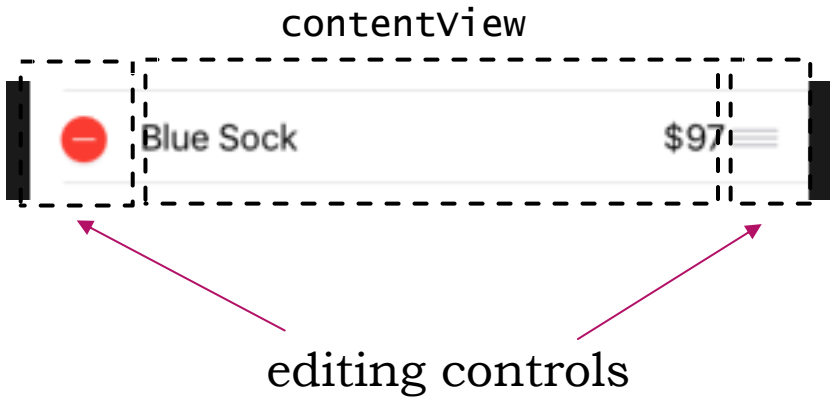




- This is sufficient for some apps
- But by subclassing `UITableViewCell`, we can create a more complex view for a row

Subclassing `UITableViewCell`

- We customize the appearance of the `UITableViewCell` by adding subviews to its `contentView`
  - and not to its cell itself
- When editing controls become visible, the `contentView` is resized to make room for them
- If we added content directly to the cell itself, then the editing controls would obscure the content



Building a Custom Cell

Design a new cell by modifying the Prototype Cell in the `UITableView` on the Storyboard

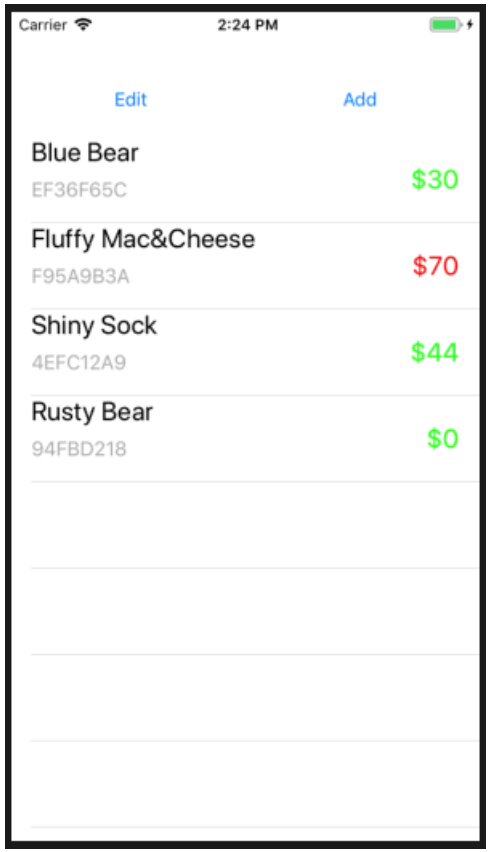
Then create a custom class to represent the cell

Then make a connection between the cell on the Storyboard and the new class

Building ItemCell

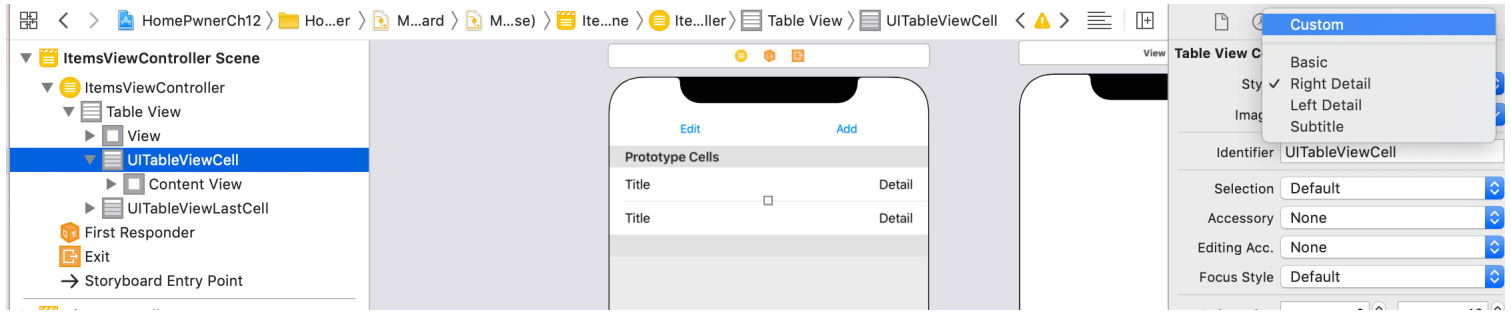
Here's the class the will represent a row in the table

```
class ItemCell: UITableViewCell {
    @IBOutlet var nameLabel: UILabel!
    @IBOutlet var serialNumberLabel: UILabel!
    @IBOutlet var valueLabel: UILabel!
}
```

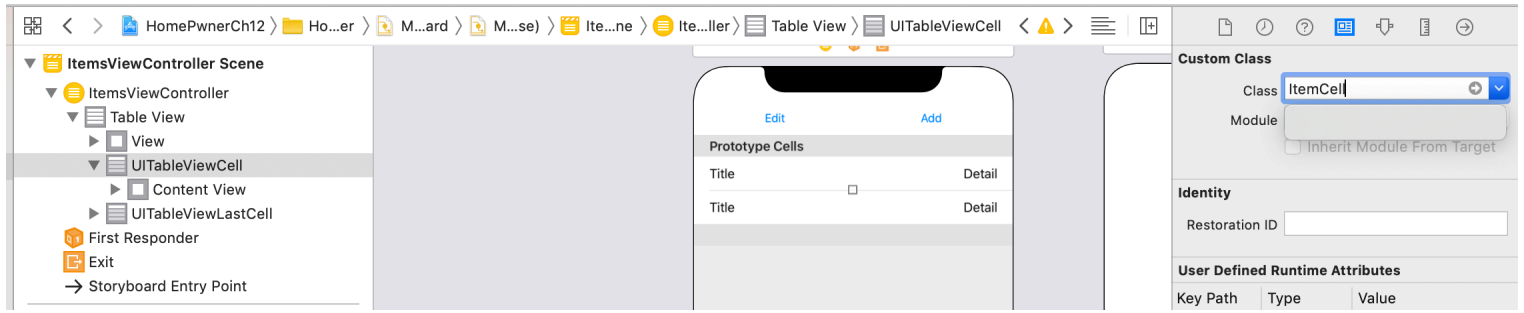


Creating a New Cell

Here are the changes on the Storyboard



change its style to Custom

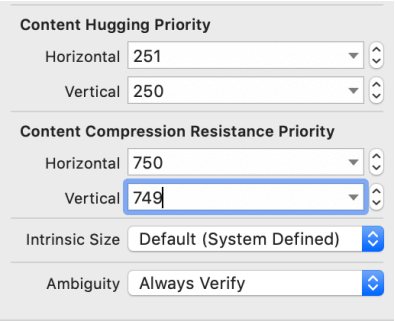


change its identifier to `ItemCell` (which is a new class we create and is a subclass of `UITableViewCell`)

Creating a New Cell

The new cell in IB, and various constraints:

- top-right label is constrained to top margin and to left margin
- leading edges of the two labels on the left are aligned
- trailing space of right-hand label is aligned with container margin
- right-hand label is centered vertically in container
- bottom left-hand label vertical content-hugging priority = 250
- bottom left-hand label vertical content compression resistance priority = 749



general gripe: constraints in IB are a pain

Outlets for the New Cell

The properties in the custom class for the cell must connect to their corresponding UI elements

```
import UIKit

class ItemCell: UITableViewCell {
    @IBOutlet var nameLabel: UILabel!
    @IBOutlet var serialNumberLabel: UILabel!
    @IBOutlet var valueLabel: UILabel!
}
```

Cell Height

We can tell the view controller to set the height of every cell to a specified value:

```
override func viewDidLoad() {
    super.viewDidLoad()
    tableView.rowHeight = 65
}
```

But the better way to do this is to let the view controller resize the cell for a row based on the contents of the cell

```
override func viewDidLoad() {
    super.viewDidLoad()
    tableView.rowHeight = UITableView.automaticDimension
    tableView.estimatedRowHeight = 65
}
```

Using the New Cell

The reuse identifier tells the `UITableViewController` what cell to use:

```
override func tableView(_ tableView: UITableView,
                        cellForRowAt indexPath: IndexPath) -> UITableViewCell {
    let item = itemStore.allItems[indexPath.row]
    let cell = tableView.dequeueReusableCell(withIdentifier: "ItemCell",
                                             for: indexPath) as! ItemCell
    cell.nameLabel.text = item.name
    cell.serialNumberLabel.text = item.serialNumber
    cell.valueLabel.text = "$\(item.valueInDollars)"
    return cell
}
```

Invalid Cell Identifier

The cell's identifier (on the Storyboard) must match the identifier in the code

- otherwise, you'll get this error:

```
2020-06-26 09:34:59.935389-0400 HomePwnerCh12[52763:56665855] *** Assertion failure in -[UITableView _dequeueReusableCellWithIdentifier:forIndexPath:usingPresentationValues:], /Library/Caches/com.apple.xbs/Sources/UIKitCore_Sim/UIKit-3920.26.113/UITableView.m:8722

2020-06-26 09:34:59.963729-0400 HomePwnerCh12[52763:56665855] *** Terminating app due to uncaught exception 'NSInternalInconsistencyException', reason: 'unable to dequeue a cell with identifier ItemCell - must register a nib or a class for the identifier or connect a prototype cell in a storyboard'
```

this is one of the many gotchas that come from not having code and storyboard in sync

Type Casting in Swift

As beautiful and elegant as Swift is, it's still necessary to do type casting is some circumstances

```
let cell = tableView.dequeueReusableCell(withIdentifier: "ItemCell",
                                                 for: indexPath) as! ItemCell
```

- `as, as?, as!` are type-casting operators in Swift
- In this case, we're forcing a downcast from a `UITableViewCell` to an `ItemCell`
- "In general, try to avoid type casting"
  - but it's necessary when we are interfacing with other libraries (such as UIKit)

Dynamic Type

Dynamic Type: a usability and accessibility feature of iOS

Users can select a type size on a real device with Settings -> Display & Brightness -> Text Size

- unfortunately, the Settings menu in the simulator doesn't have this option

Users can also pick larger text sizes from within Settings -> Accessibility

- we **can** do this in the simulator

Apps that support Dynamic Type will have their fonts scaled appropriately in response to these selections

Dynamic Type

The Dynamic Type system is centered around tags for predefined text sizes

For example, `UIFontTextStyle.title1`, `UIFontTextStyle.title2`, `UIFontTextStyle.body`

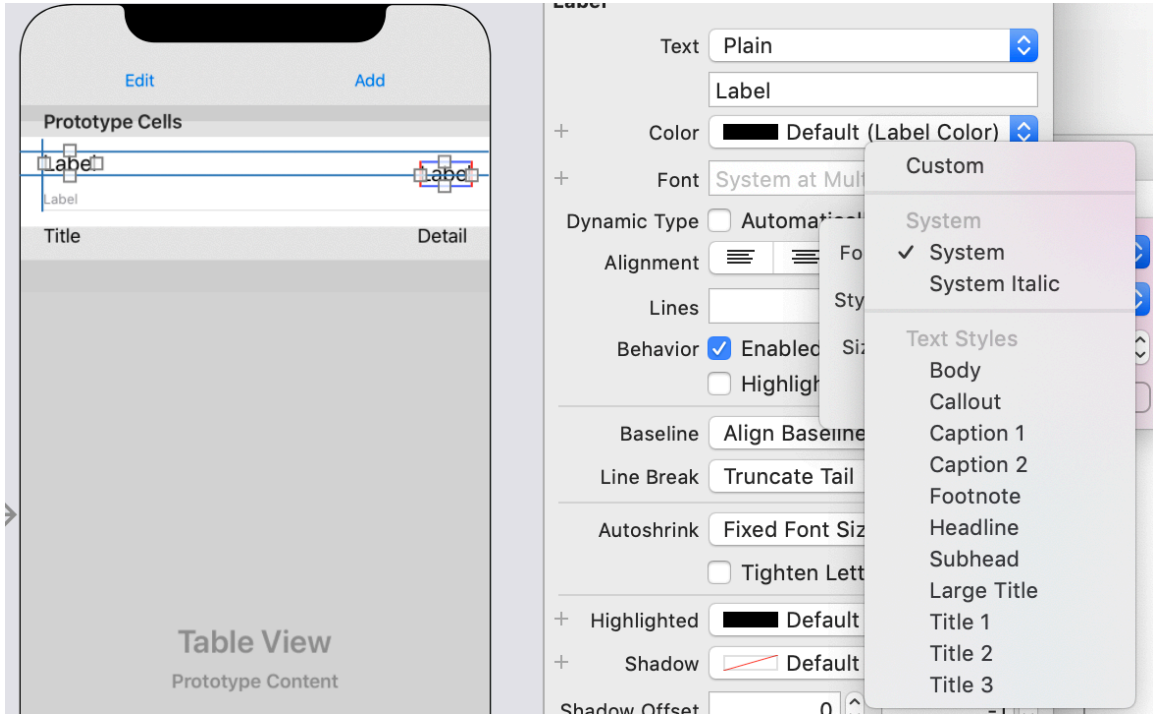
- instead of specifying fixed font sizes (e.g., 24 point)

This way, all of the labels that are `UIFontTextTyle.title1` will change uniformly when the user changes the text size

This is basic iOS UI goodness

Dynamic Type

Changing the font to one of the Dynamic Type styles



Changing Text Style

Note: the text at the bottom of p. 219 is wrong—you will see the changes if you switch back to the application

Suggested Challenge

Do the Bronze Challenge

- Update the `ItemCell` to display the `valueInDollars` in green if the value is less than 50 and in red if the value is greater than or equal to 50

This will require five lines of new code

- The question is…where to put the code?