EE443 / EE593
Mobile Application Development

Prof. Phillip De Leon

Lecture 7:
Chapter 10: The Data Model

Prof. Phillip De Leon

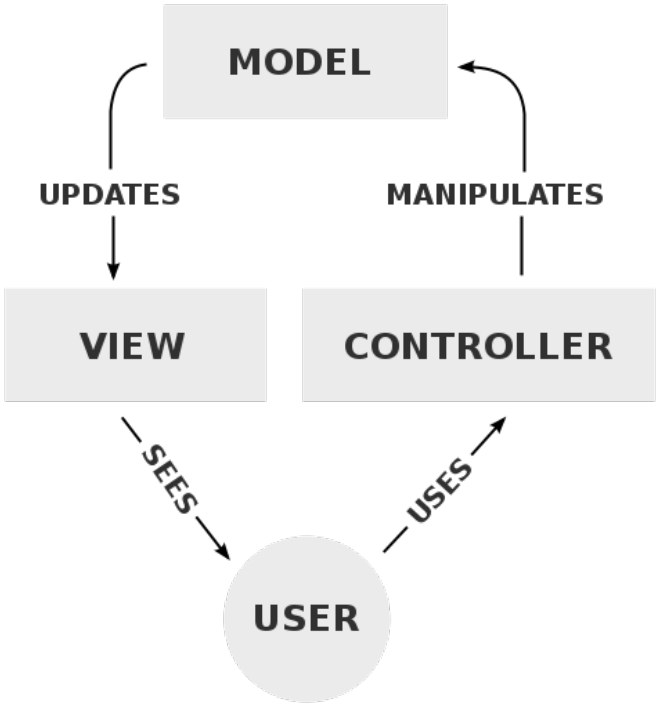
Outline

- Model-View-Controller (MVC)
- Programming To-Do List

Model-View-Controller (MVC)

- Three design patterns in iOS:

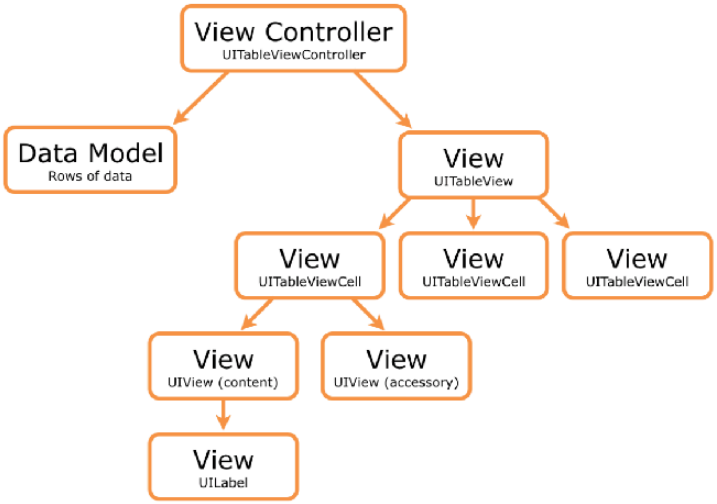
1. *Delegation* - making one object do something on behalf of another
2. *Target-Action* - connecting events such as button taps to action methods
3. *MVC* - objects are either model (contain data and data operations), view (visual part of the app), controller (connects model objects to views)



https://en.wikipedia.org/wiki/Model–view–controller

View Controllers

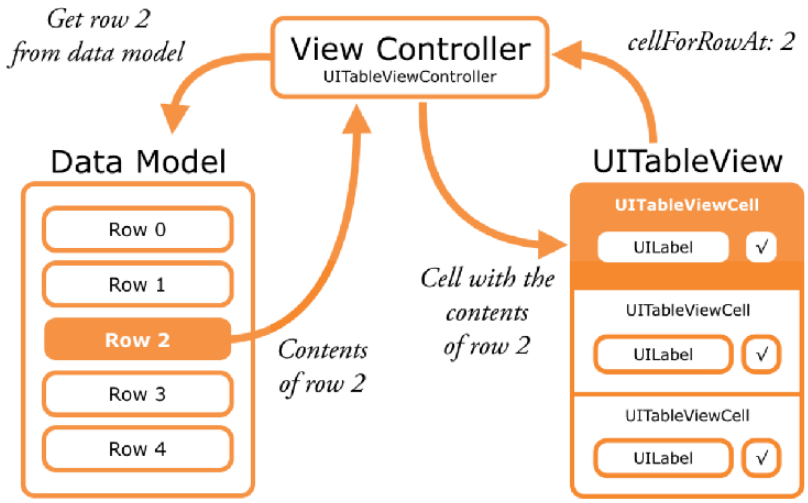
- A *view* is something you can see (button, table, label)
- A *view controller* is the bridge between data and views
- A view controller has one main view (whole screen) that contains a bunch of subviews (buttons, labels, table view cells, etc)
- Generally, a view controller handles one screen of the app. If your app has more than one screen, each of these is handled by its own view controller, i.e. in BullsEye we had a ViewController and an AboutViewController
- iOS comes with many ready-to-use view controllers



How Model-View-Controller works

Table View Controller

- Table view cells are part of the view (display data which comes from model)
- TableViewController ties rows (data) to cells (view) through implementation of table view's data source and delegate methods
- In Checklists app, each to-do items has 1) text ("Walk the dog") and 2) checkmark state



The table view controller (data source) gets the data from the model and puts it into the cells

Outline

- Model-View-Controller (MVC)
- Programming To-Do List

Programming To-Do List
Chapter 10

- Build the data model (instead of using fake data)
- Get checkmarks working properly (check and scroll—rows and cells are not synced)
- Create a custom ChecklistItem object (text and checked properties)
- Use an array to store ChecklistItem objects
- Code Cleanup

To-Do: Build the Data Model

- Use instance variables with hard-coded data (data model)
- Modify the tableView protocols to use this hard-coded data

Table View Protocols

- Review UITableViewDataSource Protocol under Documentation

1. func tableView(UITableView, numberOfRowsInSection: Int) required!
   - Tells data source to return number of rows in a given section of a table view
2. func tableView(UITableView, cellForRowAt: IndexPath) required!
   - Asks data source for a cell to insert in a particular location of table view

- Review UITableViewDelegate Protocol under Documentation

3. func tableView(UITableView, didSelectRowAt: IndexPath)
   - Tells the delegate that the specified row is now selected.

1 numberOfRowsInSection

```
override func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
    return 5
}
```

We are going to hard-code five
To-Do items (data) for now

2 cellForRowAt

```
override func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
      let cell = tableView.dequeueReusableCell(withIdentifier: "ChecklistItem", for: indexPath)
      let label = cell.viewWithTag(1000) as! UILabel
      if indexPath.row == 0 {
        label.text = row0text
      } else if indexPath.row == 1 {
        label.text = row1text
      } else if indexPath.row == 2 {
        label.text = row2text
      } else if indexPath.row == 3 {
        label.text = row3text
      } else if indexPath.row == 4 {
        label.text = row4text
      }
      configureCheckmark(for: cell, at: indexPath)  // custom function shortly
      return cell
  }
```

Use some instance vars, e.g. row0text, for To-Do text

We are going to hard-code five To-Do items (data) for now

3 didSelectRowAt

```
override func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
  if let cell = tableView.cellForRow(at: indexPath) {
    if indexPath.row == 0 {
      row0checked = !row0checked
    } else if indexPath.row == 1 {
      row1checked = !row1checked
    } else if indexPath.row == 2 {
      row2checked = !row2checked
    } else if indexPath.row == 3 {
      row3checked = !row3checked
    } else if indexPath.row == 4 {
      row4checked = !row4checked
    }
    configureCheckmark(for: cell, at: indexPath)  // custom function shortly
  }
  tableView.deselectRow(at: indexPath, animated: true)
}
```

Use some instance vars, e.g. row0checked, for To-Do checkmark state

We are going to hard-code five To-Do items (data) for now

To-Do: Get Checkmarks Working Properly

- Build a method to with checkmark toggling logic

configureCheckmark()

```
func configureCheckmark(for cell: UITableViewCell, at indexPath: IndexPath) {
  var isChecked = false
  if indexPath.row == 0 {
    isChecked = row0checked
  } else if indexPath.row == 1 {
    isChecked = row1checked
  } else if indexPath.row == 2 {
    isChecked = row2checked
  } else if indexPath.row == 3 {
    isChecked = row3checked
  } else if indexPath.row == 4 {
    isChecked = row4checked
  }
  if isChecked {
    cell.accessoryType = .checkmark
  } else {
    cell.accessoryType = .none
  }
}
```

To-Do: Create a Custom ChecklistItem Object

- Define a new class for our data object with two properties (text and checked)
- Remove the instance variables and create 5 ChecklistItem objects
- Fix if-else to use the ChecklistItem object properties instead of text and checked ivars

```
class ChecklistItem {
    var text = ""
    var checked = false
}
```

```
var row0item: ChecklistItem
var row1item: ChecklistItem
var row2item: ChecklistItem
var row3item: ChecklistItem
var row4item: ChecklistItem
```

Initialize the Objects

- Write an init() to initialize the objects (init called when object comes into existence)
- When ViewController is instantiated at startup, init?(coder) is called
- The initializer is a special type of method (which is why it doesn't start with the word func)

```
required init?(coder aDecoder: NSCoder) {
  row0item = ChecklistItem()
  row0item.text = "Walk the dog"
  row0item.checked = false
  row1item = ChecklistItem()
  row1item.text = "Brush my teeth"
  row1item.checked = true
  row2item = ChecklistItem()
  row2item.text = "Learn iOS development"
  row2item.checked = true
  row3item = ChecklistItem()
  row3item.text = "Soccer practice"
  row3item.checked = false
  row4item = ChecklistItem()
  row4item.text = "Eat ice cream"
  row4item.checked = true
  super.init(coder: aDecoder)
}
```

To-Do: Use an Array of ChecklistItem Objects

- Remove the instance vars and add an items array of type ChecklistItem
- When ViewController is instantiated at startup, init?(coder) is called
- Instantiate the array object
- Add the ChecklistItem object to the array

```
var items: [ChecklistItem]
...
items = [ChecklistItem]()
...
items.append(row0item)
```

Fix Code to Use Array

- Get a reference to the item and use this in place of the if-else logic

```
tableView:numberOfRowsInSection
     return items.count

tableView:cellForRowAt
    let item = items[indexPath.row] // get the reference

tableView:didSelectRowAt
    let item = items[indexPath.row] // get the reference

configureCheckmark()
    let item = items[indexPath.row] // get the reference
```

To-Do: Code Clean Up

- configureCheckmark() can now take the ChecklistItem directly instead of an indexPath (either works!) — replace the at: indexPath with with: item
- move code from tableView:cellForRowAt to configureText()

```
let label = cell.viewWithTag(1000) as! UILabel
label.text = item.text
```

- implement a toggleChecked method in CheckitemList Class (objects should change their own state)