060010815:
iOS Application Development

Collection View and Data Persistence

SAVING, LOADING, AND APPLICATION STATES

Archiving

- Model objects are responsible for holding on to the data that the user manipulates.
- View objects reflect that data, and controllers are responsible for keeping the views and the model objects in sync.
- Therefore, we can say, saving and loading “data” means saving and loading model objects.

Archiving

- Archiving an object involves recording all of its properties and saving them to the filesystem.
- *Unarchiving* re-creates the object from that data.

Archiving

- Classes whose instances need to be archived and unarchived must conform to the **NSCoding** protocol and implement its two required methods, **encode(with:)** and **init(coder:)**.

```
protocol NSCoding {
    func encode(with aCoder: NSCoder)
    init?(coder aDecoder: NSCoder)
}
```


Archiving

- Open Homepwner.xcodeproj and add **NSCoding** protocol declaration in Item.swift.

- class Item: NSObject**, NSCoding** {

Archiving

- In Item.swift, implement **encode(with:)** to add the names and values of the item's properties to the stream.

```
func encode(with aCoder: NSCoder) {
    aCoder.encode(name, forKey: "name")
    aCoder.encode(dateCreated, forKey: "dateCreated")
    aCoder.encode(itemKey, forKey: "itemKey")
    aCoder.encode(serialNumber, forKey: "serialNumber")

    aCoder.encode(valueInDollars, forKey: "valueInDollars")
}
```


Archiving

- In Item.swift, implement **init(coder:)**.

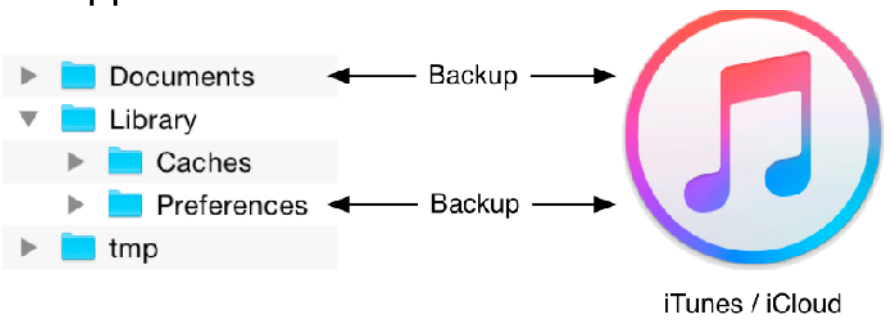
```
required init(coder aDecoder: NSCoder) {
    name = aDecoder.decodeObject(forKey: "name") as! String
    dateCreated = aDecoder.decodeObject(forKey: "dateCreated") as! Date
    itemKey = aDecoder.decodeObject(forKey: "itemKey") as! String
    serialNumber = aDecoder.decodeObject(forKey: "serialNumber") as! String?

    valueInDollars = aDecoder.decodeInteger(forKey: "valueInDollars")

    super.init()
}
```

Application Sandbox

- Every iOS application has its own *application sandbox*.
- An application sandbox is a directory on the filesystem that is barricaded from the rest of the filesystem.
- iOS application must stay in its sandbox, and no other application can access its sandbox.

Application Sandbox

- Application sandbox



Application Sandbox

- Application sandbox
  - Documents/
    - This directory is where application writes data
    - It is backed up when the device is synchronized with iTunes or iCloud.
  - Library/Caches/
    - It does not get backed up when the device is synchronized with iTunes or iCloud.
    - If the device is very low on disk space, the system may delete the contents of this directory.

Application Sandbox

- Application sandbox
  - Library/Preferences/
    - the Settings application looks for application preferences here
    - handled automatically by the class **NSUserDefaults**
    - backed up when the device is synchronized with iTunes or iCloud
  - tmp/
    - The OS may purge files in this directory when your application is not running.
    - But it is recommended to explicitly remove files from this directory when you no longer need them.

Constructing a file URL

- Implement a new property in ItemStore.swift to store this URL.

```
var allItems = [Item]()
let itemArchiveURL: URL = {
    let documentsDirectories =
        FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
    let documentDirectory = documentsDirectories.first!
    return documentDirectory.appendingPathComponent("items.archive")
}()
```

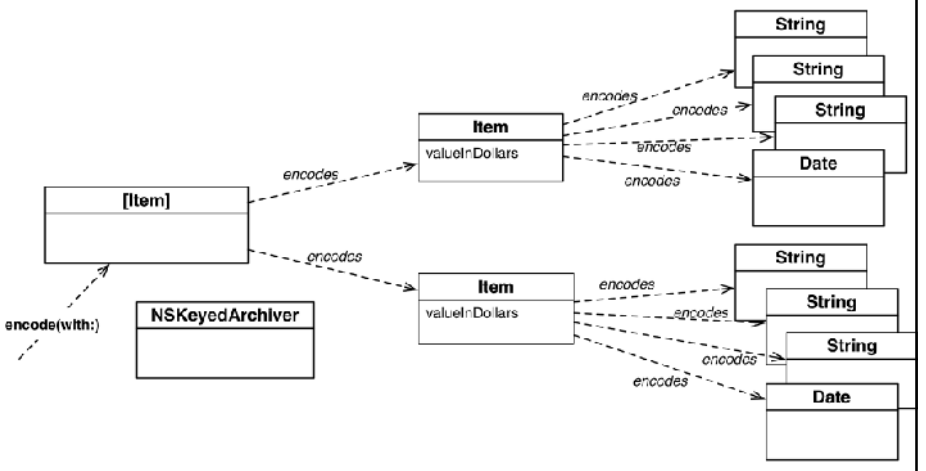

NSKeyedArchiver and NSKeyedUnarchiver

- In ItemStore.swift, implement a new method that calls **archiveRootObject(_:toFile:)**

```
func saveChanges() -> Bool {
    print("Saving items to: \(itemArchiveURL.path)")
    return NSKeyedArchiver.archiveRootObject(allItems, toFile: itemArchiveURL.path)
}
```

NSKeyedArchiver and NSKeyedUnarchiver

- Archiving the all Items array



NSKeyedArchiver and NSKeyedUnarchiver

- Open AppDelegate.swift and add a property

```
class AppDelegate: UIResponder, UIApplicationDelegate {

    var window: UIWindow?
    let itemStore = ItemStore()


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

        // Create an ItemStore
        let itemStore = ItemStore()
```

NSKeyedArchiver and NSKeyedUnarchiver

- In AppDelegate.swift

```
func applicationDidEnterBackground(_ application: UIApplication) {
    let success = itemStore.saveChanges()
    if (success) {
        print("Saved all of the Items")
    } else {
        print("Could not save any of the Items")
    }
}
```

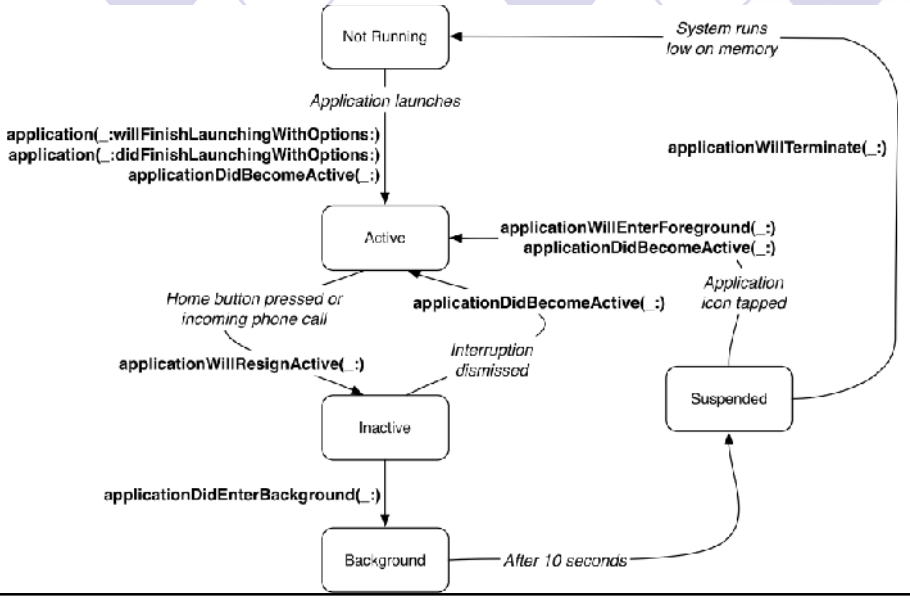

Loading files

- In ItemStore.swift,

```
init() {
    if let archivedItems =
        NSKeyedUnarchiver.unarchiveObject(withFile: itemArchiveURL.path) as? [Item] {
        allItems = archivedItems
    }
}
```

Application States and Transitions

- States of a typical application



Application States and Transitions

- Application states

State	Visible	Receives Events	Executes Code
Not Running	no	no	no
Active	yes	yes	yes
Inactive	mostly	no	yes
Background	no	no	yes
Suspended	no	no	no

Writing to the Filesystem with Data

- How to save item image???

- In ImageStore.swift

```
func imageURL(forKey key: String) -> URL {

    let documentsDirectories =
        FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)
    let documentDirectory = documentsDirectories.first!

    return documentDirectory.appendingPathComponent(key)
}
```


Writing to the Filesystem with Data

- In ImageStore.swift,

```
func setImage(_ image: UIImage, forKey key: String) {
    cache.setObject(image, forKey: key as NSString)

    // Create full URL for image
    let url = imageURL(forKey: key)

    // Turn image into JPEG data
    if let data = UIImageJPEGRepresentation(image, 0.5) {
        // Write it to full URL
        let _ = try? data.write(to: url, options: [.atomic])
    }
}
```

Load image from filesystem

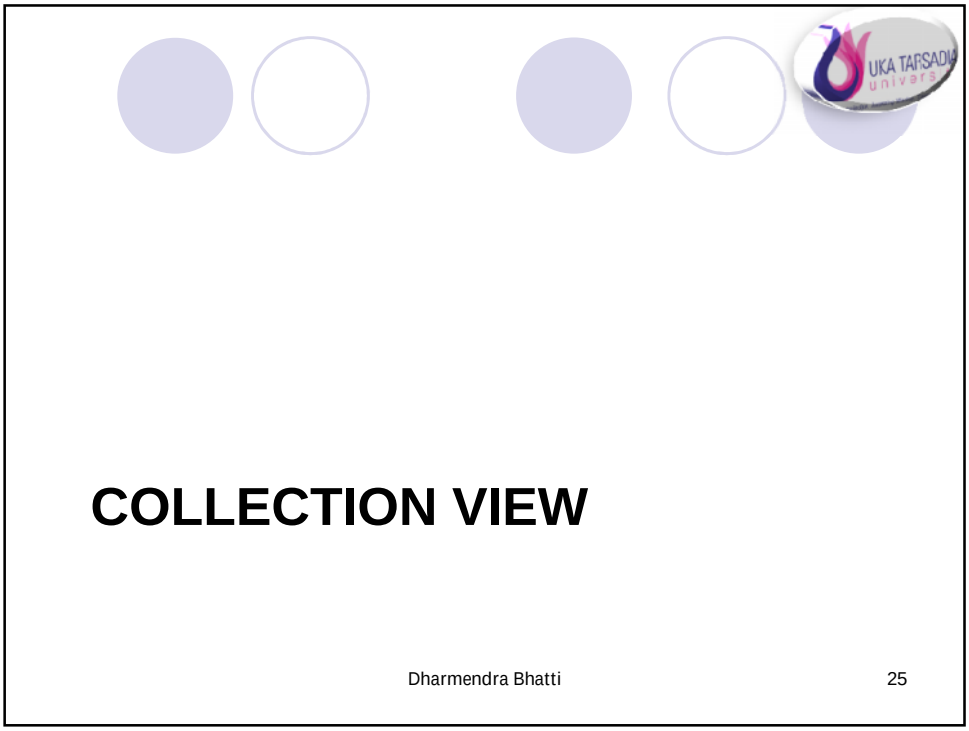
- In ImageStore.swift,

```
func image(forKey key: String) -> UIImage? {
    return cache.object(forKey: key as NSString)

    if let existingImage = cache.object(forKey: key as NSString) {
        return existingImage
    }

    let url = imageURL(forKey: key)
    guard let imageFromDisk = UIImage(contentsOfFile: url.path) else {
        return nil
    }

    cache.setObject(imageFromDisk, forKey: key as NSString)
    return imageFromDisk
}
```

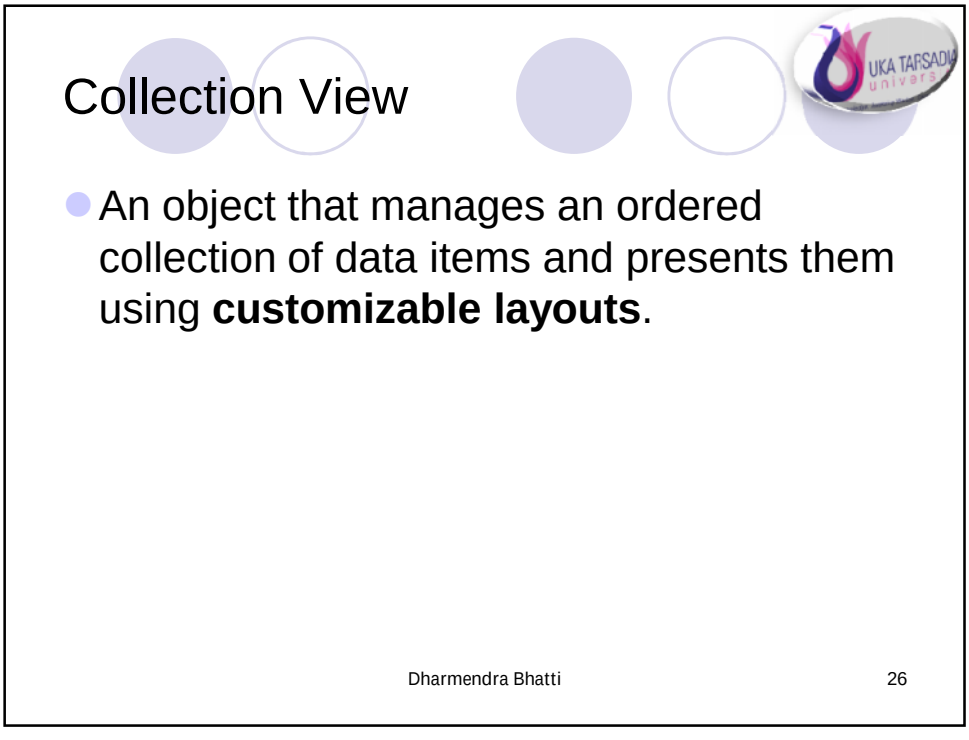

Delete image

- In ImageStore.swift,

```
func deleteImage(forKey key: String) {
    cache.removeObject(forKey: key as NSString)

    let url = imageURL(forKey: key)
    do {
        try FileManager.default.removeItem(at: url)
    } catch {
        print("Error removing the image from disk: \(error)")
    }
}
```



COLLECTION VIEW



Collection View

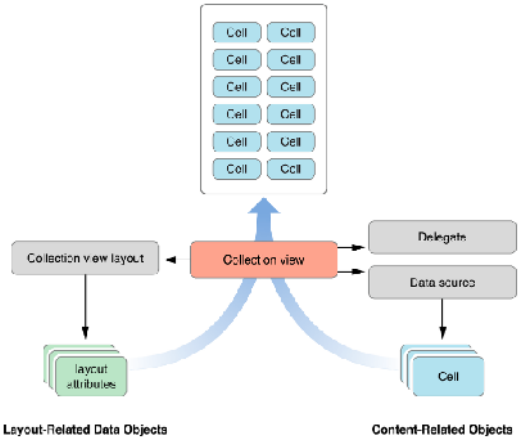
- An object that manages an ordered collection of data items and presents them using **customizable layouts**.

Collection View

- Collection of data items in customizable layouts
- Collection of cells
- Each cell consists of item/data

Collection View

- Like Table View



Collection View

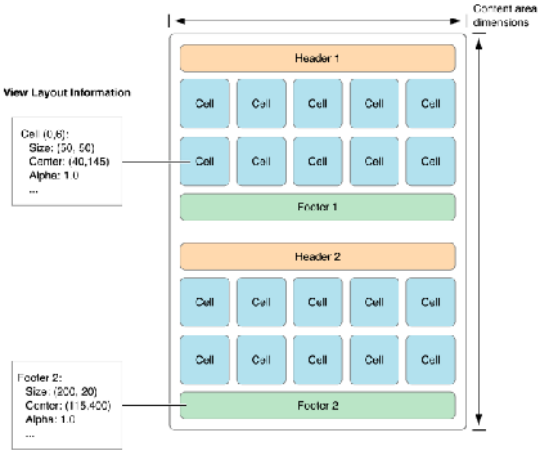
- The collection view gets information about the cells to display from its data source
- The data source and delegate objects are used to manage the content, including the selection and highlighting of cells.

Collection View

- The layout object is responsible for deciding where those cells belong and for sending that information to the collection view in the form of one or more layout attribute objects.

Collection View

- The layout object provides layout metrics



31

Creating DEMO - CollectionController

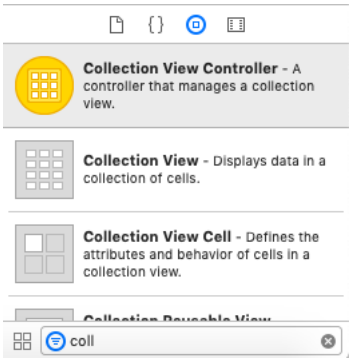
- Create a new project
  - Select *single view application*
  - Choose a name, etc. Make sure the language is *Swift* and it's for the *iPhone*
  - Leave unchecked: *use core data*
  - Choose a place to save
    - Do *not* need to use git

Creating DEMO - CollectionController

- Main Story Board => Delete ViewController
- Project Navigator => Delete ViewController.swift

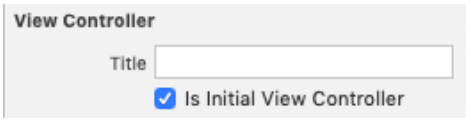
Creating DEMO - CollectionController

- Create CollectionViewController by dragging it from object library.



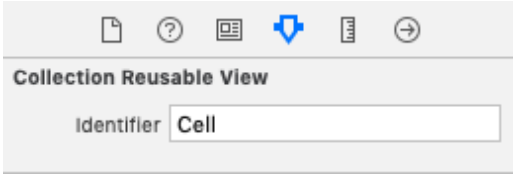
Creating DEMO - CollectionController

- Select CollectionViewController => Attribute Inspector => check the "Is Initial View Controller" checkbox.



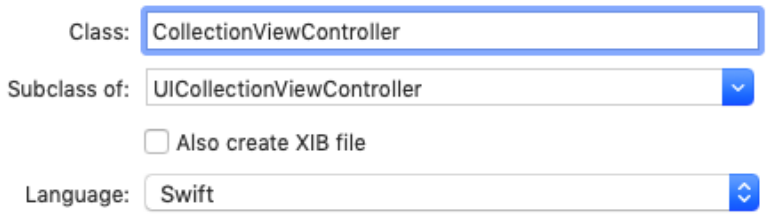
Creating DEMO - CollectionController

- Select the Collection View Cell and go to the Attribute inspector.
- In the Collection Reusable View section set the Identifier to "Cell".



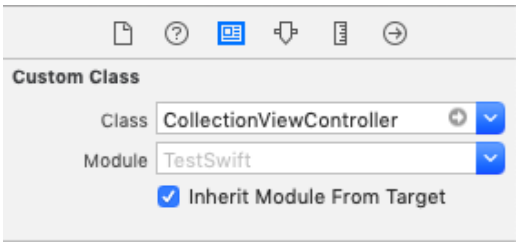
Creating DEMO - CollectionController

- File => New => File… => Cocoa Touch Class



Creating DEMO - CollectionController

- Select Collection View Controller => Identity Inspector => change Custom Class to CollectionViewController



Creating DEMO - CollectionController

- CollectionViewController.swift

```
var cellColor = true
```


Creating DEMO - CollectionController

- CollectionViewController.swift

```
override func numberOfSections(in collectionView: UICollectionView) -> Int {
    return 1
}
override func collectionView(_ collectionView: UICollectionView,
                             numberOfItemsInSection section: Int) -> Int {
    return 100
}
```

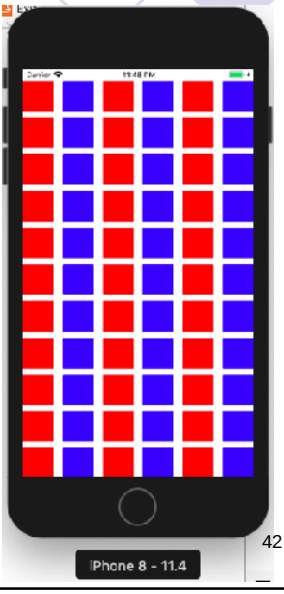
Creating DEMO - CollectionController

- CollectionViewController.swift

```
override func collectionView(_ collectionView:
    UICollectionView, cellForItemAt indexPath: IndexPath)
    -> UICollectionViewCell {
    let cell = collectionView.dequeueReusableCell(
        withReuseIdentifier: reuseIdentifier, for: indexPath)

    // Configure the cell
    cell.backgroundColor = cellColor ? UIColor.red : UIColor.blue
    cellColor = !cellColor

    return cell
}
```


Creating DEMO - CollectionController

- Build and run the project



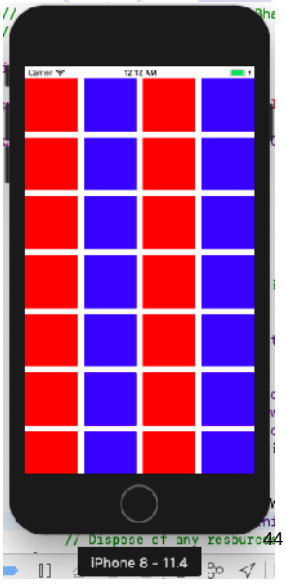
Creating DEMO - CollectionController

- CollectionViewController.swift => viewDidLoad( )

```
let width = (view.frame.width - 30)/4
let layout = collectionView?.collectionViewLayout
    as! UICollectionViewFlowLayout
layout.itemSize = CGSize(width: width, height: width)
```


Creating DEMO - CollectionController

- Build and run the project



Creating DEMO - StudentCollection

- 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

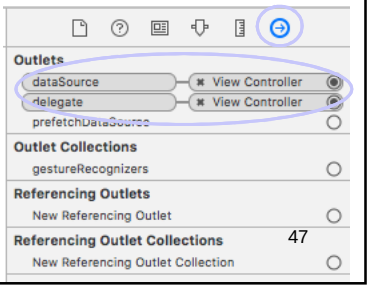
Creating the view

- Click on the Main.Storyboard to show IB
- Find a *Collection View* in the library and drag to the view.
- Using Auto Layout, **Add 4 Constraints**

Creating the view

- Open the hierarchical view by clicking on the Document Outline button
- Click on the Collection 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 outline

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



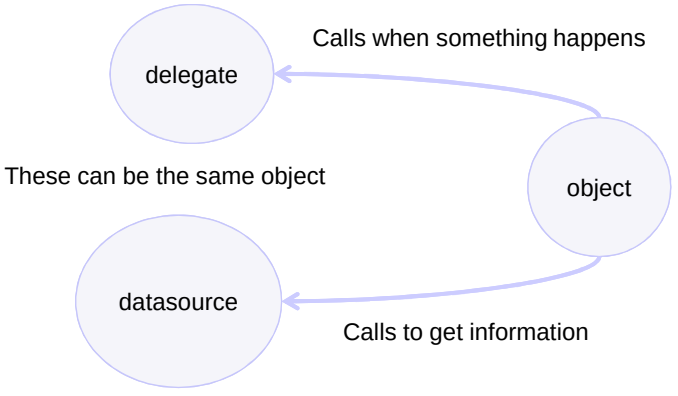
Delegates & Datasources

- **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



Demo - StudentCollection

- Main.storyboard => CollectionView => select cell => Size Inspector
- Set
  - Size = custom
  - Width = 140
  - Height = 180

Demo - StudentCollection

- Add in to the collection view cell
  - ImageView
    - Set width 120, height 120
  - Label
    - Change system font to 13
  - Label
    - Change system font to 13

Demo - StudentCollection

- Create new swift file “StudentCell”
- import UIKit
- class StudentCell: UICollectionViewCell {
-
- }

Demo - StudentCollection

- Select collection view cell => Identity Inspector => set class = “StudentCell”
- Select collection view cell => Attribute Inspector => set identifier = “StudentCell”

Demo - StudentCollection

- Create IBOutlets for ImageView and both labels
- Go to Assistant Editor
  - We need Main.storyboard and StudentCell.swift
  - From top, select manual, project name, StudentCell.swift
- Create outlets
  - Ctrl+Drag ImageView to StudentCell.swift “studentImage”
  - Ctrl+Drag label to StudentCell.swift “enrollment”
  - Ctrl+Drag label to StudentCell.swift “name”

Demo - StudentCollection

- class StudentCell: UICollectionViewCell {
- @IBOutlet var studnetImage: UIImageView!
- @IBOutlet var enrollment: UILabel!
- @IBOutlet var name: UILabel!
- }

Demo - StudentCollection

- Create students information as dictionary in ViewController.swift

```
var students = [
    ["enrollment":"201406100110079","name":"Jaiminkumar","imageName":"201406100110079.png"],
    ["enrollment":"201406100110136","name":"Abhishek","imageName":"201406100110136.png"],
    ["enrollment":"201506100110001","name":"Khyati","imageName":"201506100110001.png"],
    ["enrollment":"201506100110005","name":"Unnati","imageName":"201506100110005.png"],
    ["enrollment":"201506100110007","name":"Vinanti","imageName":"201506100110007.png"],
    ["enrollment":"201506100110008","name":"Amisha","imageName":"201506100110008.png"],
    ["enrollment":"201506100110009","name":"Aenykumari","imageName":"201506100110009.png"],
    ["enrollment":"201506100110011","name":"Raj","imageName":"201506100110011.png"],
    ["enrollment":"201506100110013","name":"Darshan","imageName":"201506100110013.png"],
    ["enrollment":"201506100110016","name":"Hetvi","imageName":"201506100110016.png"],
    ["enrollment":"201506100110018","name":"Kajalben","imageName":"201506100110018.png"],
    ["enrollment":"201506100110019","name":"Mansi","imageName":"201506100110019.png"],
    ["enrollment":"201506100110023","name":"Preet","imageName":"201506100110023.png"],
    ["enrollment":"201506100110025","name":"Raghav","imageName":"201506100110025.png"],
    ["enrollment":"201506100110027","name":"Jigisha","imageName":"201506100110027.png"],
    ["enrollment":"201506100110029","name":"Jugal","imageName":"201506100110029.png"],
    ["enrollment":"201506100110030","name":"Raj","imageName":"201506100110030.png"]
]
```

Demo - StudentCollection

- In ViewController.swift,
- Conform to UICollectionViewDatasource and UICollectionViewDelegate
- class ViewController: UIViewController, **UICollectionViewDataSource**, **UICollectionViewDelegate** {

Demo - StudentCollection

- Add following functions in to ViewController.swift

```
func collectionView(_ collectionView: UICollectionView,
                    numberOfItemsInSection section: Int) -> Int {
    return students.count
}
```


Demo - StudentCollection

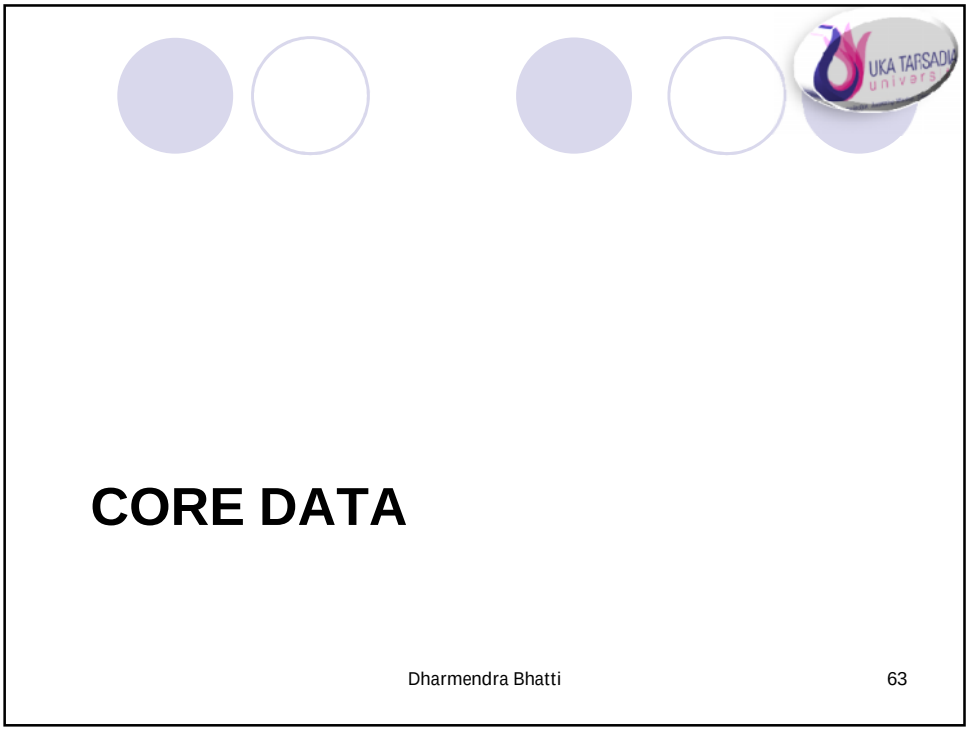
- func **collectionView**(_ collectionView: UICollectionView, **cellForItemAt** indexPath: IndexPath) -> UICollectionViewCell {
- let cell:StudentCell = collectionView.dequeueReusableCell(withReuseIdentifier: "StudentCell", for: indexPath) as! StudentCell
-
- cell.enrollment.text = self.students[indexPath.row]["enrollment"]
- cell.name.text = self.students[indexPath.row]["name"]
- cell.studnetImage.image = UIImage(named: self.students[indexPath.row]["imageName"]!)
-
- return cell
- }

Demo - StudentCollection

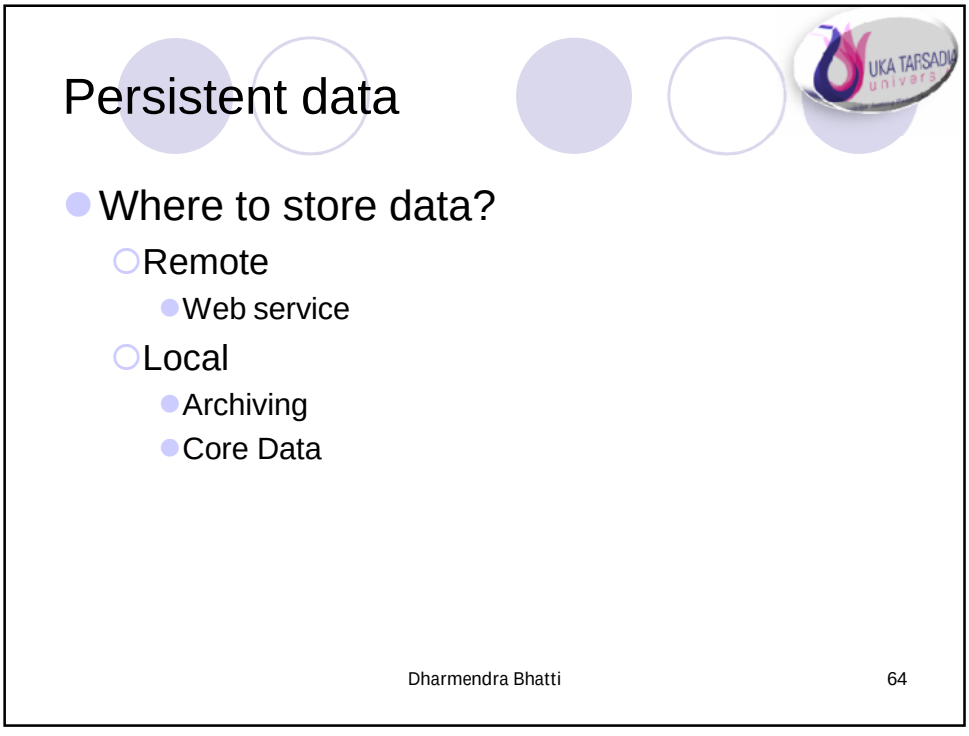
- Add images to project
- Select Assets.xcassets and drag required images there

Demo - StudentCollection

- Build and run the application.



CORE DATA



Persistent data

- Where to store data?
  - Remote
    - Web service
  - Local
    - Archiving
    - Core Data

Persistent data

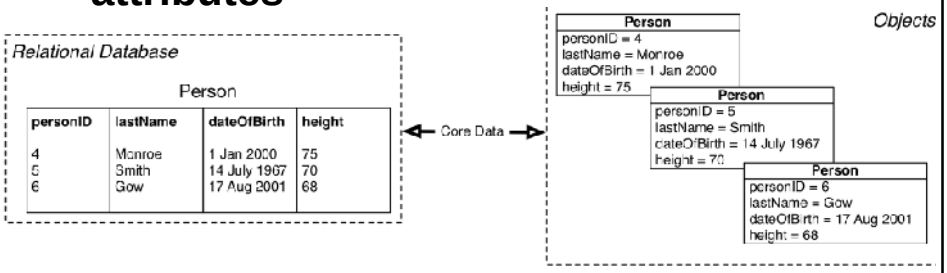
Archiving	Core Data
To access anything in the archive, you must unarchive the entire file.	Can fetch a subset of the stored objects.
To save any changes, you must rewrite the entire file.	Can update just desired part of the file.
all-or-nothing nature	Incremental fetching, updating, deleting, and inserting

Core Data

- Core Data is a framework that lets you express what your model objects are and how they are related to one another.
- We just tell Core Data *what* needs storing and let it work out *how* to store it.
  - (Don't worry about database schema and foreign keys)

Core Data

- “table” or “type” is called “**entity**” in Core Data
- “column” or “properties” are called “**attributes**”



Core Data

- By default, Core Data uses a SQLite database as the persistent store

Core Data

- Core Data provides **on-disk persistence**, which means data will be accessible even after terminating your app or shutting down your device.
- While in-memory persistence will only save data as long as app is in memory, either in the foreground or in the background.

Core Data

- Xcode comes with a powerful **Data Model editor**, which you can use to create your **managed object model**.
- A managed object model is made up of **entities**, **attributes** and **relationships**
- An **entity** is a class definition in Core Data.

Core Data

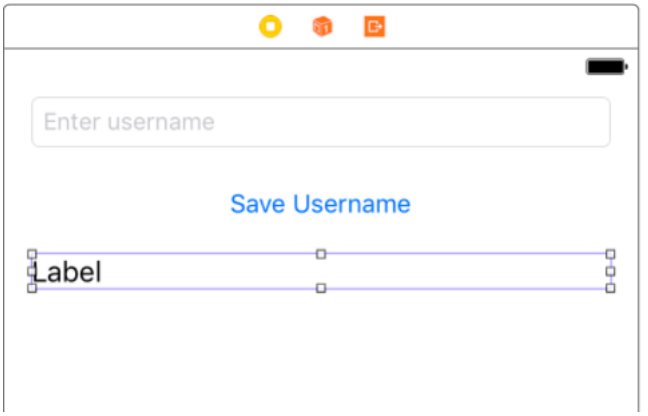
- An **attribute** is a piece of information attached to an entity.
- A **relationship** is a link between multiple entities.

Core Data

- An NSManagedObject is a run-time representation of a Core Data entity.
  - You can read and write to its attributes using **Key-Value Coding**.
- You need an NSManagedObjectContext to save() or fetch(_:) data to and from Core Data.

RememberMe Demo

- Create a new project “RememberMe Demo”

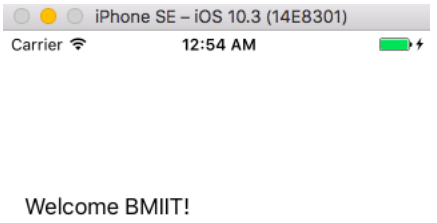


RememberMe Demo

- For the first run, app will ask username

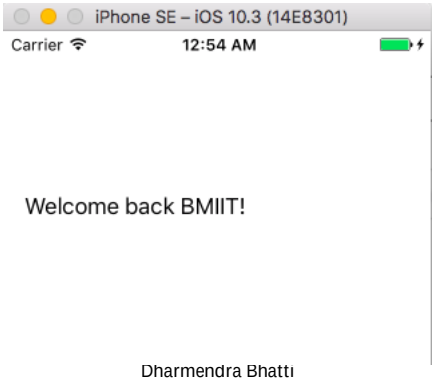
RememberMe Demo

- On tap of “Save username”



RememberMe Demo

- For all subsequent run, app will greet “Welcome back”



Creating DEMO - RememberMe

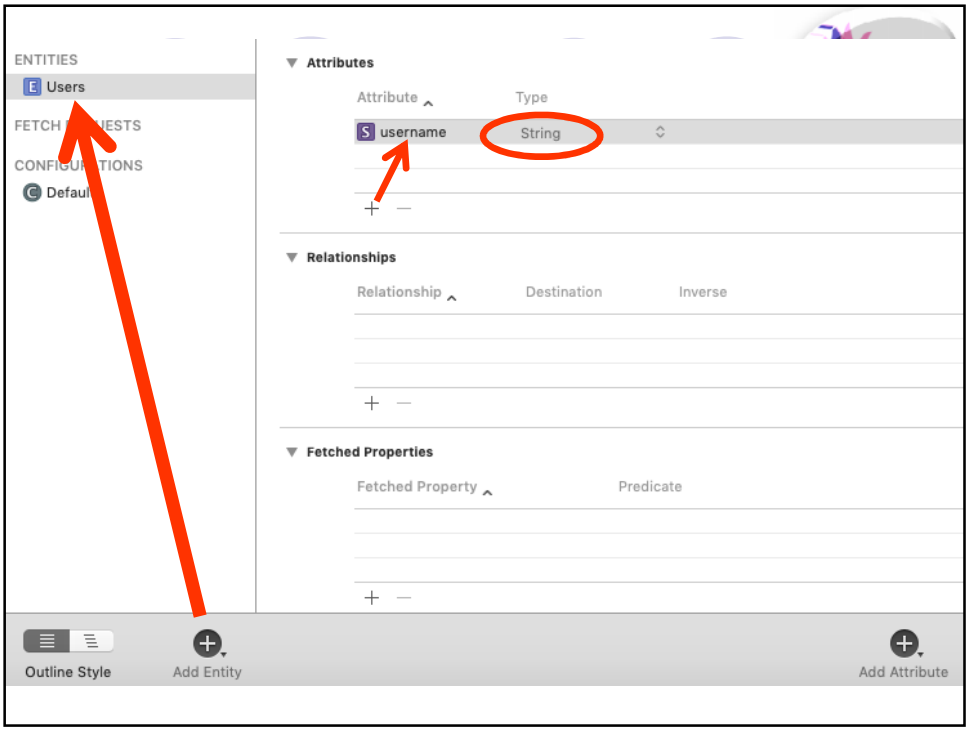
- Create a new project
  - Select *single view application*
  - Choose a name, etc. Make sure the language is *Swift* and it's for the *iPhone*
  - **Check: *use core data***
  - Choose a place to save
    - Do *not* need to use git

RememberMe Demo

- There are two notable changes
  - The new file RememberMe.xcdatamodeld
  - The AppDelegate.swift file with Core Data Stack code

RememberMe Demo

- Click on RememberMe.xcdatamodeld file
  - click on “Add Entity” (+) button and name the entity as “Users”
  - click on “Add Attribute” (+) button and name the attribute as “username” and type “String”



RememberMe Demo

- Main Story Board => View Controller => ADD a TextField, a Button, a Label
- Apply Auto Layout
- Create 3 @IBOutlets and an @IBAction (see next two slides).

RememberMe Demo

- Import UIKit
- **Import CoreDate**
- class ViewController: UIViewController {
- @IBOutlet var textField: UITextField!
- @IBOutlet var label: UILabel!
- @IBOutlet var saveButton: UIButton!

RememberMe Demo

```
class ViewController: UIViewController {

 @IBAction func buttonTapped(_ sender: UIButton) {
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
            let context = appDelegate.persistentContainer.viewContext
        let newUser = NSEntityDescription.insertNewObject(forEntityName: "Users",
into: context)
        newUser.setValue(textField.text, forKey: "name")
          do {
           try context.save()
           textField.alpha = 0
           label.alpha = 1
           saveButton.alpha = 0
           label.text = "Welcome " + textField.text! + "!"
        } catch {
           print("Save failed")
            }
       }
```


RememberMe Demo

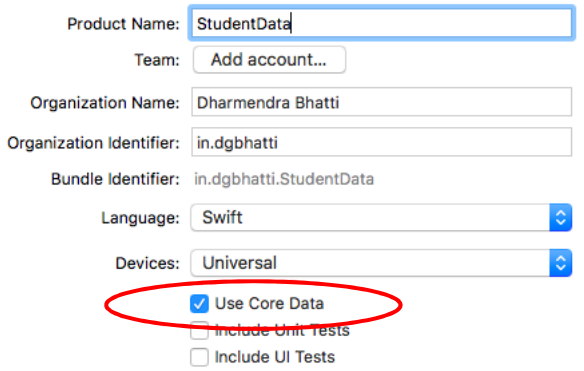
```
class ViewController: UIViewController {
 override func viewDidLoad() {
        super.viewDidLoad()
        let appDelegate = UIApplication.shared.delegate as! AppDelegate
        let context = appDelegate.persistentContainer.viewContext
        let request = NSFetchRequest<NSFetchRequestResult>(entityName: "Users")
        do {
           let results = try context.fetch(request)
           for result in results as! [NSManagedObject] {
              if let username = result.value(forKey: "name") as? String {
                 textField.alpha = 0
                 label.alpha = 1
                 saveButton.alpha = 0
                 label.text = "Welcome back " + username + "!"
                       }
                 }
        } catch {
           print("Read failed")
            }
       }
```

RememberMe Demo

- Build and run the application.

Demo - StudentData

- Create a NEW project “StudentData” and make sure “Use Core Data” is checked



Demo - StudentData

- Checking the **Use Core Data** box will cause Xcode to generate boilerplate code for what's known as an NSPersistentContainer in **AppDelegate.swift**.
- The **NSPersistentContainer** consists of a set of objects that facilitate **saving** and **retrieving** information from Core Data.

Demo - StudentData

- Main.storyboard => Select view controller => Editor menu => Embed In => Navigation Controller
- Drag a *Table View* from the object library into the view controller
- Use autolayout to cover the entire view.

Demo - StudentData

- Drag a *Bar Button Item* and place it on the view controller's navigation bar.
- Select the bar button item and change its system item to *Add*.
- Make the view controller the table view's data source (Ctrl+drag from table view to view controller and select dataSource)

Demo - StudentData

- From Navigator area, select StudentData.xcdatamodeld
- Click on *Add Entity* on the lower-left to create a new entity.
- Double-click the new entity and change its name to "Student"

Demo - StudentData

- Add an attribute to Student object
- Set the new attribute's name to, "*name*" and change its type to *String*
- Open ViewController.swift and add
  - import CoreData

Demo - StudentData

- Ctrl-drag from the table view onto *ViewController.swift*, inside the class definition to create an IBOutlet "tabelView"
- @IBOutlet weak var tableView: UITableView!

Demo - StudentData

- Ctrl-drag from the *Add* button into *ViewController.swift* and select action

- @IBAction func addName(_ sender: UIBarButtonItem) {
-
- }

Demo - StudentData

- Create model for the table view

- var students: [NSManagedObject] = [ ]

Demo - StudentData

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

  title = ”Student List"
  tableView.register(UITableViewCell.self,
                     forCellReuseIdentifier: "Cell")
}
```


Extension

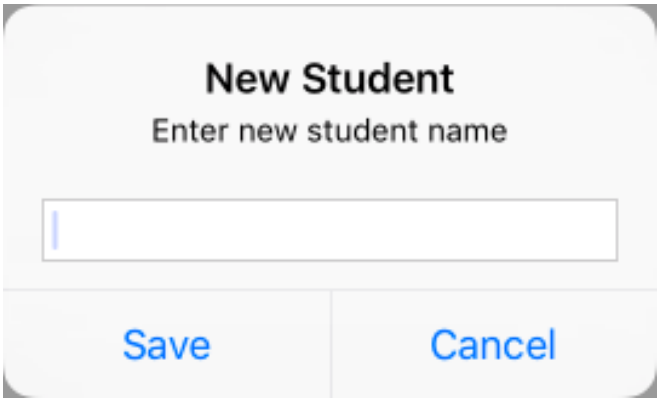
- *Extensions* add new functionality to an existing class, structure, enumeration, or protocol type.
- This includes the ability to extend types for which you do not have access to the original source code (known as ***retroactive modeling***).

Demo - StudentData

```
extension ViewController: UITableViewDataSource {

  func tableView(_ tab leView: UITableView,
                 numberOfRowsInSection section: Int) -> Int {
    return students.count
  }

  func tableView(_ tableView: UITableView,
                 cellForRowAt indexPath: IndexPath)
                 -> UITableViewCell {
    let student = students[indexPath.row]
    let cell =
      tableView.dequeueReusableCell(withIdentifier: "Cell",
                                    for: indexPath)
    cell.textLabel?.text = student.value(forKeyPath: "name") as? String
    return cell
  }
}
```


Alert Controller

- Use alert controller to get NEW name



Demo - StudentData

```
@IBAction func addName(_ sender: AnyObject) {
  let alert = UIAlertController(title: "New Student",
                                message: "Enter new student name",
                                preferredStyle: .alert)
  let saveAction = UIAlertAction(title: "Save",
                                 style: .default) {
    [unowned self] action in

    guard let textField = alert.textFields?.first,
      let nameToSave = textField.text else {
        return
    }
…
```


Demo - StudentData

```
@IBAction func addName(_ sender: AnyObject) {
…
    self.save(name: nameToSave)
    self.tableView.reloadData()
  }
let cancelAction = UIAlertAction(title: "Cancel",
                                 style: .default)
alert.addTextField()

  alert.addAction(saveAction)
  alert.addAction(cancelAction)

  present(alert, animated: true)
}
```

Demo - StudentData

```
func save(name: String) {
  guard let appDelegate =
    UIApplication.shared.delegate as? AppDelegate else {
    return
  }
let managedContext =
    appDelegate.persistentContainer.viewContext
let entity =
    NSEntityDescription.entity(forEntityName: ”Student",
                               in: managedContext)!
let student = NSManagedObject(entity: entity,
                               insertInto: managedContext)
student.setValue(name, forKeyPath: "name”)
…
```


Demo - StudentData

```
func save(name: String) {
…
do {
    try managedContext.save()
    students.append(student)
  } catch let error as NSError {
    print("Could not save. \(error), \(error.userInfo)")
  }
}
```

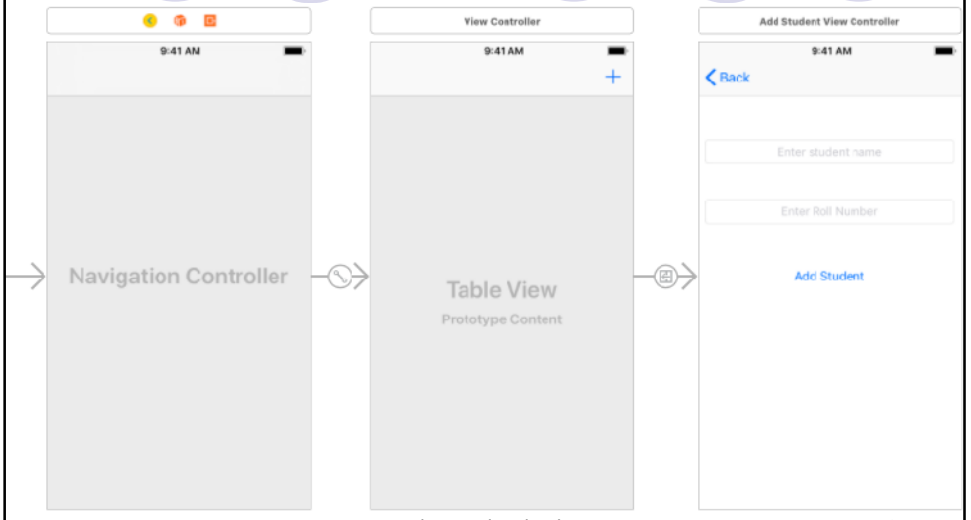
Demo - StudentData

```
override func viewWillAppear(_ animated: Bool) {
  super.viewWillAppear(animated)

  guard let appDelegate =
    UIApplication.shared.delegate as? AppDelegate else {
      return
  }
  let managedContext =
    appDelegate.persistentContainer.viewContext
  let fetchRequest =
    NSFetchRequest<NSManagedObject>(entityName: "Student")

  do {
    students = try managedContext.fetch(fetchRequest)
  } catch let error as NSError {
    print("Could not fetch. \(error), \(error.userInfo)")
  }
}
```


Demo - StudentData

- Build and run the application

Demo2 – StudentData2



Demo2 – StudentData2

- Create a new project StudentData2
  - Navigation Controller
  - TableView
  - Bar Button – Add

Demo – StudentData2

- From Navigator area, select StudentData.xcdatamodeld
- Click on *Add Entity* on the lower-left to create a new entity.
- Double-click the new entity and change its name to “Student”

Demo – StudentData2

- Add an attribute to Student object
- Set the new attribute’s name to, “*name*” and change its type to *String*

Demo – StudentData2

- Add another attribute to Student object
- Set the new attribute's name to, "*enrollment*" and change its type to *String*

Demo – StudentData2

- Select "Student" Entity => Click on "Editor" menu => Create NSManagedObject Subclass…
- Two new swift files generated
  - Student+CoreDataClass.swift
  - Student+CoreDataProperties.swift

Demo – StudentData2

- ViewController

```
class ViewController: UIViewController, UITableViewDataSource {

    @IBOutlet var tableView: UITableView!

    let managedObjectContext = (UIApplication.shared.delegate as!
        AppDelegate).persistentContainer.viewContext

    var students: [Student] = []
```


Demo – StudentData2

- ViewController

```
override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a n
    tableView.dataSource = self
    title = "Student List"
}

override func viewWillAppear(_ animated: Bool) {
    getData()
    tableView.reloadData()
}

func getData() {
    do {
        students = try managedObjectContext.fetch(Student.fetchRequest())
    } catch {
        print("Fetching failed...")
    }
}
```

Demo – StudentData2

- ViewController

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

func tableView(_ tableView: UITableView, cellForRowAt
    indexPath: IndexPath) -> UITableViewCell {
    let cell = UITableViewCell(style: .subtitle, reuseIdentifier: "Cell")
    let student = students[indexPath.row]
    if let myName = student.name, let rollno = student.enrollment {
        cell.textLabel?.text = myName
        cell.detailTextLabel?.text = rollno
    }
    return cell
}
```

Demo – StudentData2

- ViewController

```
func tableView(_ tableView: UITableView, commit
    editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
    if editingStyle == .delete {
        let student = students[indexPath.row]

        managedObjectContext.delete(student)

        (UIApplication.shared.delegate as! AppDelegate).saveContext()

        do {
            students = try managedObjectContext.fetch(Student.fetchRequest())
        } catch {
            print("Fetching failed")
        }
    }
    tableView.reloadData()
}
```

Demo – StudentData2

- Main Story Board => from object library add ViewController
  - nameTextField
  - enrollmentTextField
  - Add Student Button
- File => New => File… => Cocoa Touch Class “addStudentViewController”
- Set class for newly created view controller

class AddStudentViewController …

- Create 2 outlets and 1 action

```
@IBOutlet var textField: UITextField!
@IBOutlet var enrollmentField: UITextField!

@IBAction func addStudentTapped(_ sender: UIButton) {

    if textField.text != "" {
        let managedObjectContext = (UIApplication.shared.delegate as!
            AppDelegate).persistentContainer.viewContext

        let student = Student(context: managedObjectContext)
        student.name = textField.text!
        student.enrollment = enrollmentField.text!

        (UIApplication.shared.delegate as! AppDelegate).saveContext()
    }

    let _ = navigationController?.popViewController(animated: true)
}
```

Demo – StudentData2

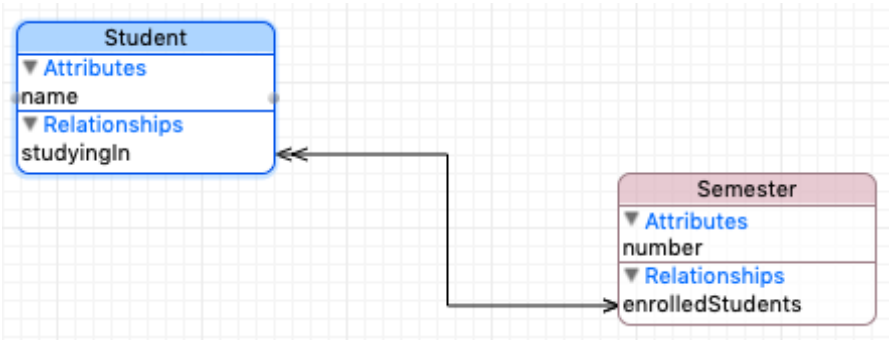
- Connect Add Bar Button to addStudentViewController with Segue
- Ctrl+Drag from Add Bar Button to second view controller and select => Action Segue => Show

Demo2 – StudentData2

- Build and run the application

Relations

- Student-Semester data model



Relations

- Delete Rules
  - No Action
    - We have a semester record that contains several student records.
    - If the semester record is deleted, the student records are not notified of this event.
    - Every student record thinks it is still associated with the deleted semester.
  - Nullify
  - Cascade
  - Deny

Relations

- Delete Rules
  - No Action
  - Nullify
    - If the delete rule of a relationship is set to **Nullify**, the destination of the relationship is nullified when the record is deleted.
    - If a semester has several students and the semester is deleted, the relationship pointing from the student to the semester is nullified.
    - This is the default delete rule and the delete rule you will find yourself using most often.
  - Cascade
  - Deny

Relations

- Delete Rules
  - No Action
  - Nullify
  - Cascade
    - Cascade is useful if the data model includes one or more dependencies.
    - For example, if an account record has a relationship to one or more user records, it may be desirable to delete every user if the account the user belongs to is deleted.
  - Deny

Relations

- Delete Rules
  - No Action
  - Nullify
  - Cascade
  - Deny
    - Instead of cascading the deletion of a record, it **prevents** the deletion of a record.
    - For example, if an account is associated with several users, the account can only be deleted if it is no longer tied to any users.
    - This configuration prevents the scenario in which users are no longer associated with an account.

Questions ???