ATLS 4120: Mobile Application Development
Week 6: Data Persistence

Data Persistence

In most apps users change or add data, and we need that data to be persistent.
Our model objects hold data and should support data persistence so you can write objects to a file and then read them back in.
The most common ways to handle data persistence in iOS are:

- Property lists
- Object archives
- SQLite3 (iOS's embedded relational database)
- Core Data (Apple's persistence framework)

We'll be using plists today but the book covers all 4. We'll also be looking at the Realm framework and Firebase next semester.

Property Lists

A property list is a simple data file in XML that stores item types and values. They use a key to retrieve the value, just as a Dictionary does. Property lists can have Boolean, Data, Date, Number, and String node types to hold individual pieces of data, as well as Arrays or Dictionaries to store collections of nodes.
All of our apps have an Info.plist file in Supporting Files.
Property lists can be created using the Property List Editor application (*/Developer/Applications/Utilities/Property List Editor.app*) or directly in Xcode.
We are going to use a property list to store our app's data.

Only certain objects can be stored in property lists and then written to a file.

- Array or NSArray
- Dictionary or NSDictionary
- NSData
- String or NSString
- (and the mutable versions of the above)
- NSNumber
- Date or NSDate

If you can build your data model from just these objects, you can use property lists to save and load your data. This is ok for simple data models. Otherwise use another method.

Sandbox

Your app sees the iOS file system like a normal UNIX file system
Every app gets its own /Documents directory which is referred to as its sandbox
Your app can only read and write from that directory for the following reasons:

- Security (so no one else can damage your application)
- Privacy (so no other applications can view your application's data)
- Cleanup (when you delete an application, everything its ever written goes with it)

To find your sandbox in the Finder go into your home directory and go to
Library/Developer/CoreSimulator/Devices/*Device UDID*/data/Containers/Data/Application
(The Library option is hidden so if you don't see the Library folder in your home directory Go | Go to Folder | Library hold down the alt key, or hold the Option key Go | Library)

Each app has it's own folder (names are the globally unique identifiers(GUIDs) generated by xcode
Each app has subdirectories

- Documents-app sandbox to store its data
- Library-user preferences settings
- Tmp-temp files (not backed up into iTunes)

The same file structure exists on devices. You can connect your device to your computer and in Window | Devices and Simulators select your device and you will see Installed Apps. Select an app and click the gear and Show Container to see the sandbox contents of that app.

The FileManager class is part of the Foundation framework and provides and interface to the file system. It enables you to perform many generic file-system operations such as locating, creating, copying, and moving files and directories.

- You can use URL or String for file location but URL is preferred
- We will use the urls(for:in:) method to locate the document directory. This method returns an array but on iOS there's only one document directory so we can just use the first item in the array

Codable

Swift 4 made it much easier to convert external data such as JSON and plists into internal representations of the data using the Encodable and Decodable protocols. These protocols will let you automatically encode and decode data into class or struct instances in your code. To support both encoding and decoding, you can use the Codable typealias, which combines the Encodable and Decodable protocols. This process is known as making your types codable.

- Types that are Codable include standard library types like String, Int, and Double; and Foundation types like Date, Data, and URL. Array, Dictionary, and Optional also conform to Codable when they contain codable types.
- Adopt Codable to the inheritance list of your class/struct (: Codable)
- The class/struct property names MUST match the key names of the items in the property list/JSON file
- Use the PropertyListEncoder and PropertyListDecoder for plists
    - decode(_:from:) will decode the data from the file. It must match the structure of the type
    - encode(_:) creates a property list of the value you pass it
- Use the JSONEncoder and JSONDecoder for JSON data

Communication

In iOS there are four common patterns for objects to communicate

1. Target-Action: a single object calls a single method when a single event occurs
    - ie buttons
2. Delegation: an object responds to numerous methods to modify or add behavior
    - text fields, table views, etc that have delegate methods
3. Notification: Register an object to be notified when an event occurs
    - Sets up how to handle when an event fires
4. Key-Value Observing (KVO): register to be one of many objects notified when single property of another object changes.
    - used for archiving

Notifications

A notification is a callback mechanism that can inform multiple objects when an event occurs.

- NotificationCenter manages the notification process

- Objects register for the notifications they're interested in
- Notification senders post notifications to a notification center
- The notification center notifies any objects registered for that notification

Data Persistence

We'll add on to our Favorites app and use a plist to make our data persistent.
ViewController.swift
Define a constant for our data file

```
let filename = "favs.plist"
```

Now we write a method that will return the url to a given file.

```
    func dataFileURL(_ filename:String) -> URL? {
        //returns an array of URLs for the document directory in the
user's home directory
        let urls = FileManager.default.urls(for:.documentDirectory,
in: .userDomainMask)
        var url : URL?
        //append the file name to the first item in the array which
is the document directory
        url = urls.first?.appendingPathComponent(filename)
        //return the URL of the data file or nil if it does not exist
        return url
    }
```

FileManager.SearchPathDirectory.documentDirectory can be shortened to .documentDirectory (value from the SearchPathDirectory enum)

FileManager.SearchPathDomainMask.userDomainMask can be shortened to . userDomainMask (value from the SearchPathDomainMask enum). On iOS this maps to the user's home directory.

On iOS, there is always only one Documents directory for each application, so we can safely assume that exactly one object will be returned and can be accessed as the first item of the returned array.

Update viewDidLoad so we see if favs.plist exists and if it does we use that.

```
    override func viewDidLoad() {
        //url of data file
        let fileURL = dataFileURL(filename)

        //if the data file exists, use it
        if FileManager.default.fileExists(atPath: (fileURL?.path)!){
            let url = fileURL!
            do {
                //creates a data buffer with the contents of the
plist
                let data = try Data(contentsOf: url)
                //create an instance of PropertyListDecoder
                let decoder = PropertyListDecoder()
                //decode the data using the structure of the Favorite
class
```

```
                user = try decoder.decode(Favorite.self, from: data)
                //assign data to textfields
                bookLabel.text=user.favBook
                authorLabel.text=user.favAuthor
            } catch {
                print("no file")
            }
          }
        else {
            print("file does not exist")
        }
```

Then our application needs to save its data before the application is terminated or sent to the background, so we'll use the UIApplicationWillResignActiveNotification notification. This notification is posted whenever an app is no longer the one with which the user is interacting. This includes when the user quits the application and (in iOS 4 and later) when the application is pushed to the background. (add after the else statement in viewDidLoad())

```
        //application instance
        let app = UIApplication.shared
        //subscribe to the UIApplicationWillResignActiveNotification
notification
        NotificationCenter.default.addObserver(self, selector:
#selector(self.applicationWillResignActive(_:)), name:
Notification.Name.UIApplicationWillResignActive, object: app)
```

Now we'll create the notification method applicationWillResignActive (_:)

```
    //called when the UIApplicationWillResignActiveNotification
notification is posted
    //all notification methods take a single NSNotification instance
as their argument
    @objc func applicationWillResignActive(_ notification:
Notification){
        //url of data file
        let fileURL = dataFileURL(filename)
        //create an instance of PropertyListEncoder
        let encoder = PropertyListEncoder()
        //set format type to xml
        encoder.outputFormat = .xml
        do {
            //encode the data using the structure of the Favorite
class
            let plistData = try encoder.encode(user)
            //write encoded data to the file
            try plistData.write(to: fileURL!)
        } catch {
            print("write error")
        }
    }
```

To test the app fill in the text fields and tap Done to go back to the first view. Tap the home button, then exit the simulator and run it again, it should load your data. If you just exit the simulator without pressing the home button, that's the equivalent of forcibly quitting your application. In that case, you will never receive the notification that the application is terminating, and your data will not be saved. This is just a function of running in the simulator and is not needed on a device.

Note that in testing this functionality you might need to change your plist file name often, otherwise once the file is created it will always find the file in viewDidLoad() and use it. So if you want to start fresh, use a different file name for your plist.