QUESTIONS

1. How to save Model class in NSUserDefault?



We can only store things like NSArray, NSDictionary, NSString, NSData, NSNumber, and NSDate in NSUserDefaults.

2 functions for saving people and retrieving people from UserDefaults, which are as below:

func savePeople(people:[Person]) {

let archivedObject = archivePeople(people: people)

let defaults = UserDefaults.standard

defaults.set(archivedObject, forKey: UserDefaultsPeopleKey)

defaults.synchronize()

}

func retrievePeople() -> [Person]? {

if let unarchivedObject = UserDefaults.standard.object(forKey: UserDefaultsPeopleKey) as? NSData {

return NSKeyedUnarchiver.unarchiveObject(with: unarchivedObject as Data) as? [Person]

}

return nil

}

For execution of saving values to defaults,

//MARK: CREATE USERS

let arrayPeople = [Person(firstname: "johnny", surname:"appleseed"),Person(firstname: "peter", surname: "mill")]

savePeople(people: arrayPeople)

For execution of retrieving values from defaults,

// RETRIEVE USERS IN DEFAULTS

if let retrievedPeople = retrievePeople() {

for person in retrievedPeople {

print("\(person.firstname), you have been unarchived")

}

} else {

print("Writing to UserDefaults is still broken in playgrounds")

}

(2) Describe view controller lifecycle order when another view controller starts over current view controller (Please explain)

Below is the sequence in which the view controller life cycle methods will be executed, when a segue happens from FirstViewController to SecondViewController

**SecondViewController.swift - init(coder:)**

While creating the views of your app in a Storyboard, init(coder:) is the method that gets called to instantiate your view controller and bring it to life.

**FirstViewController.swift - prepare(for:sender:)**

Notifies the view controller that a segue is about to be performed.

**SecondViewController.swift - loadView()**

This is the method that creates the view for the view controller.

**SecondViewController.swift - viewDidLoad()**

It’s only called when the view is created.

**FirstViewController.swift - viewWillDisappear**

Before the transition to the next view controller happens and the origin view controller gets removed from screen, this method gets called.

**SecondViewController.swift - viewWillAppear**

This method is for tasks that requires to be repeated every time a view controller comes on screen.

**SecondViewController.swift - viewDidAppear**

This method gets called after the view controller appears on screen.

**FirstViewController.swift - viewDidDisappear**

After a view controller gets removed from the screen, this method gets called.

(3) How to support multi-resolution and multi-language in iOS app? (Please explain)

**Multi-resolution Support**

For multi-resolution, AutoLayout and Size Classes can be used to make the app compatible with iOS devices with varying resolutions. Even, Universal apps can be made using AutoLayout and Size Classes.

**Universal Screen Design**

We now have a multitude of devices with widely varying screen sizes: the iPhone supports 3.5”, 4”, 4.7” and 5.5” displays while the iPad supports 7.9” and 9.7” displays (with a third potentially coming in 2015).

Apple has made it possible for developers to create universal apps that can easily adapt to all of the different devices and screen sizes with minimal effort.

**The Four Size Classes**

- Size classes are specified in both horizontal and vertical orientations and can also be specified as either Regular or Compact.

- We can now design more generically than in the past, so that it can change dynamically with differing screen sizes.

- The portrait orientation for iPhone 4S, 5, 5C, 5S, 6, 6 Plus and iPod touch has a Regular Vertical Size Class as well as a Compact Horizontal Size Class. When these devices (besides the iPhone 6 Plus) are rotated to a landscape orientation, they are classified as Compact for both Horizontal and Vertical Size Classes. The only device so far to have a Compact Vertical Size Class with a Regular Horizontal Size Class is the iPhone 6 Plus in landscape orientation.

**Customize Layouts for Every Orientation**

Since an app’s functionality might change between devices, developers can feel free to customize their app’s layout for every orientation of different screen sizes.

**Multi-language support**

For multi-language support, localisation is to be implemented

**Localizing the Storyboard**

When we add a localization, Xcode scans through the base storyboard, extracts those textual items to be localized and put them into the strings file. The visible strings of storyboards such as label, title of the navigation bar and button title are put into the strings file. All entries are in key/value pairs. The first part of key is the object ID of the UI item. We can find the object ID of the UI object under Identity Inspector.

**Localizing Image**

- Select the image in the project navigator and click on Localize button

- Switch back to the finder and locate the project directory. You’ll find two folders: en.lproj and fr.lproj. Both folders are automatically generated by Xcode for localization. The en.lproj folder stores resource files for English localization, while fr.lproj folder is for French localization.

**Localizing Dynamic Strings**

* We use the NSLocalizedString macro to get the string. The NSLocalizedString macro fetches a localized string from the Localizable.strings file for the current localization.

**Localizing the App Name**

- When we create the French localization, Xcode already generates the French version of InfoPlist.strings. You can put key/value pairs in the InfoPlist.string file to override the values stored in Info.plist file.

"CFBundleDisplayName" = “Librairie";

(4) Explain the error handling and exception management

In Swift, errors are represented by values of types that conform to the Error protocol.

Error Handling is just additional way to write an else-if statement to not only deal with the error messages but also respond after them in a separate block.

Swift enumerations are particularly well suited to modeling a group of related error conditions, with associated values allowing for additional information about the nature of an error to be communicated. Create an enum with the possible errors. The enum should confirm to the error protocol.

enum ErrorsToThrow: Error {

case fileNotFound

case fileNotReadable

case fileSizeIsTooHigh

}

We can design Throwable Function, which “throws” an error which will be "caught" and handled in a separate block with do-try.

There are four ways to handle errors in Swift:

— You can propagate the error from a function to the code that calls that function.

— Handle the error using a do-catch statement.

— Handle the error as an optional value (try?).

— Assert that the error will not occur (try!).