Project Description

This series of coding challenges is designed to prepare you for the most common causes of friction between Objective-C and Swift.

Why this project?

When you build an app that employs both of the languages of iOS you want to know that when a variable is passed from Objective-C to Swift, Swift knows what to do with it. With this practice set you can develop good habits for dealing with variables exported by Objective-C in forms that are difficult for Swift to handle, namely, nil values and unspecified types.

What will I learn?

Common problems in interoperability between Objective-C and Swift, including:

- Nullability annotations
- Error handling
- Careful use of the AnyObject class
- How to make your Swift API visible to Objective-C

Why is this project meaningful to my career?

An iOS developer who is able to use APIs written in both Objective-C and Swift is a real asset to any iOS engineering team. Handling interoperability between Swift and Objective-C is a relatively new challenge and not a trivial one. Mastering this skill will set you apart from other job candidates.

To access the project files for this problem set go to the ios-nanodegree-obj-c-student repo and look in the "Problem Set" folder.

1) Annotate the PlaneTicket project. In the plane ticket project you'll see two classes, `PlaneTicket` and `Passenger`. Add nullability annotations to each. Remember you can check your work by inspecting the generated header.

2) Add nullability annotations to the classes `Sweatshirt` and `WashingMachine` in the Sweatshirt project.

3) Inspect the Animals_Swift project. Use what you've learned about avoiding the hazards of `AnyObject` to fix the two runtime errors.

4) Inspect the Toolbox project. Use what you've learned about avoiding the hazards of `AnyObject` to fix the runtime error.

5) The Regift Cocoapod uses the following enum to represent a Regift error:

```
public enum RegiftError: String, ErrorType {
    case DestinationNotFound = "The temp file destination could not be created or f
    case AddFrameToDestination = "An error occurred when adding a frame to the dest
    case DestinationFinalize = "An error occurred when finalizing the destination"
}
```

Edit this enum so that it can be consumed successfully by an Objective-C class. Submit your answer in a swift file.

6) Inspect the Guitar_SwiftSpecific project. You'll notice that there are 7 compiler errors. Fix these compiler errors by translating the Swift-specific code into code that can be consumed by Objective-C. Hint: start with the Swift structs.

7) Open up the GuitarString project and write some Objective-C code in the main.m file that will process an error from the method, `pluck()`. You will need to follow these steps:

- Create an instance of the `GuitarString` class
- Create an `NSError` object
- With the `GuitarString` you created, call the method `pluck(velocity: Float)` - remember that it will have a different name once it has been translated to Objective-C.
- Check if an error was returned
- Log the error

8) Open up the RPS_MixedLanguage project and you'll see that it looks very much like the RockPaperScissors_CommandLine project, but the `RPSTurn` class has been written in Swift. Right now the code in the main.m file won't compile. Adjust the project settings and import statements so that the Swift and Objective-C classes in this project can work together. Remember to:

- Ensure that any .m file that uses `RPSTurn` has imported the required header file
- Ensure that any .h file that uses `RPSTurn` has forward-declared the class

Note that since you cannot use a Swift enum in an Objective-C header file, like RPSController.h, you will need to modify the method `throwDown()`. Experiment with different strategies and remember to visit the forum for advice if you get stuck.

PROJECT SPECIFICATION

Common Interoperability Challenges

Nullability Annotations, Problems 1&2

CRITERIA	MEETS SPECIFICATIONS
Use of `_Nullable` and `_Nonnull` annotations with properties.	All properties in the `Sweatshirt` and `PlaneTicket` projects are annotated correctly, with either `_Nullable` or `_Nonnull`.
Use of `_Nullable` and `_Nonnull` with return types and parameters.	All return types and parameters in the `Sweatshirt` and `PlaneTicket` projects are annotated correctly using either `_Nullable` or `_Nonnull`.

AnyObject, Preventing "unrecognized selector" errors, Problems 3&4

CRITERIA	MEETS SPECIFICATIONS
Students are carefully handling objects of type `AnyObject` and avoiding "unrecognized selector" errors	The two runtime errors in `Animals_Swift` and one runtime error in `Toolbox` are avoided using either optional chaining or optional casting.

Making Swift APIs visible to Objective-C, Problems 5&6

CRITERIA	MEETS SPECIFICATIONS

Students are able to write a Swift enum so that it can be consumed by Objective-C	The `RegiftError` enum is of type `NSInteger`, conforms to the `ErrorType` protocol, no longer contains strings, and is marked with the attribute `@objc`.
	`/// Errors thrown by Regift`
	`@objc public enum RegiftError: Int, ErrorType {`
	`  case DestinationNotFound`
	`  case AddFrameToDestination`
	`  case DestinationFinalize`
	`}`

Students are able to convert Swift-specific constructs into code that can be consumed by Objective-C, i.e. convert structs into classes	In the `Guitar_SwiftSpecific` project, all 7 errors have been fixed, the code compiles and runs

Processing Swift-style errors from Objective-C

CRITERIA	MEETS SPECIFICATIONS
Students are able to process an error passed from Swift to Objective-C	Students have followed these steps in the `GuitarString` project: 1. Create an instance of the `GuitarString` class 2. Create an `NSError` object 3. With the `GuitarString` you created, call the method `pluck(velocity: Float)` - remember that it will have a different name once it has been translated to Objective-C. 4. Check if an error was returned 5. Log the error

Setting up a Mixed Language Project

CRITERIA	MEETS SPECIFICATIONS
Students are able to correctly configure a mixed language project	Students have followed these steps in the `RPS_MixedLanguage` project: 1. “Build Settings” indicate that the project contains embedded swift code 2. All `.m` files that use `RPSTurn` have imported the required header file 3. All `.h` files that use `RPSTurn` have forward-declared the class

Student FAQ