README.md

RPNCalculator-ObjC

Introduction

The goal of this project is to continue to familarize yourself with the Objective-C language by recreating your RPN Calculator in ObjC. It will help you practice the concepts learned today, including primitives vs. classes, `NSNumber`, and mutability.

After completing the lesson material and this project, you should be able to:

- explain the difference between a class and a primitive, along with common examples of each
- understand and briefly explain the concept of pointers
- use NSNumber to convert between primitive numbers and objects
- understand and explain the difference between immutable and mutable classes
- use copy/mutableCopy to convert from immutable to mutable and vice versa

You are welcome and encouraged to use your existing Swift solution to this project as inspiration and help when implementing this project. However, instead of using a custom framework and importing it into your project, you will be creating all the logic in your own project.

Instructions

You will need to create your own Xcode project and repository. Commit regularly as you complete the requirements in this project.

Part 1 - Storyboard

1. Delete the existing ViewController files in your project and in the storyboard.
2. Drag out a new `UIViewController` and add the necessary buttons and textField for your RPN Calculator (Hint: Use the tag property for the numbers rather than creating actions for all of them).
    - You can reference the layout of the ViewController in your other project to make sure you have all the buttons laid out correctly
    - You could even copy the Main.storyboard from the Swift version and simply change the class of the view controller.
3. Make sure your view controller is the initial view controller.

Part 2 - Model/Model Controller

1. Create a new file called `XXXStack` (XXX = your class prefix).

- `.h` file: - Declare the methods to push, pop and peek the stack. - Declare an `initWith` initalizer that takes an `NSArray`.
- `.m` file: - Create a private `NSMutableArray` property called `values` that will be used every time your methods are called. - Implement the methods you declared in your `.h` file to alter your `values` array. - Inside of your `initWith...` initializer assign a `mutableCopy` of your `array` property to your internal `values` array.

2. Create another file called `XXXCalculator`.

- `.h` file: - Create an enum for the operations that will be calculated using `typedef NS_ENUM(rawType, nameOfEnum) {};`. Inside the curly braces you will just need to separate each item with a comma. - Declare three methods `- (void)pushNumber:(double)value;`, `- (void)applyOperator:(YourEnumType)operator;` and `- (void)clear;`. - Also declare a `topValue` property that will be a computed property in your `.m` file.
- `.m` file: - Every Calculator should have one `Stack` and only that calculator should know about it's `Stack`. Create a property `XXXStack *stack` in this file that we will use to implement the methods. - Implement the methods in this file by accessing the methods on your `Stack` model.

3. Create a new file called `XXXDigitAccumulator`. This will be used to accumulate digits as they're entered, then convert the list of digits into a number.

- `.h` file - Your `XXXDigitAccumulator` class should have three methods and one property: - `- addDigitWithNumericValue: - -addDecimalPoint - -clear - value` - a readonly `double` property
- `.m` file - Implement the digit accumulator. Use the Swift version for inspiration. Note that the Swift version uses an enum with associated values for digits, which you can't do directly in Objective-C. You'll have to come up with another solution! **Note:** For now, for simplicity, you can ignore the error handling that the Swift version does. Ignore repeated decimal digits instead of throwing an error, and assume that values will be in the range 0-9.

Part 3 - View Controller

Implement the view controller. Use the Swift version for inspiration. The implementation here will be nearly identical except that it will be in Objective-C.

Go Further

If you finish early or want to push yourself, here are a few additional features you can implement:

- Split the logic out into a framework as we did in Swift
- Implement error handling. What should happen if the user enters more than one decimal digit? Note that `@throws` is not available in Objective-C. Objective-C uses a different mechanism for error handling, which we'll learn about in a future lesson. Without knowing that, how would you solve this?