LP 2: Assignment – iOS Application in Swift

Compound Interest Calculator (with custom classes)

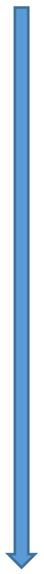
Using custom functions and classes, write an *iOS Application in Swift using Xcode* that grabs and validates input data from several views and calculates the compound interest that would be earned based on the input data. This should be done in *several parts (tasks)*:

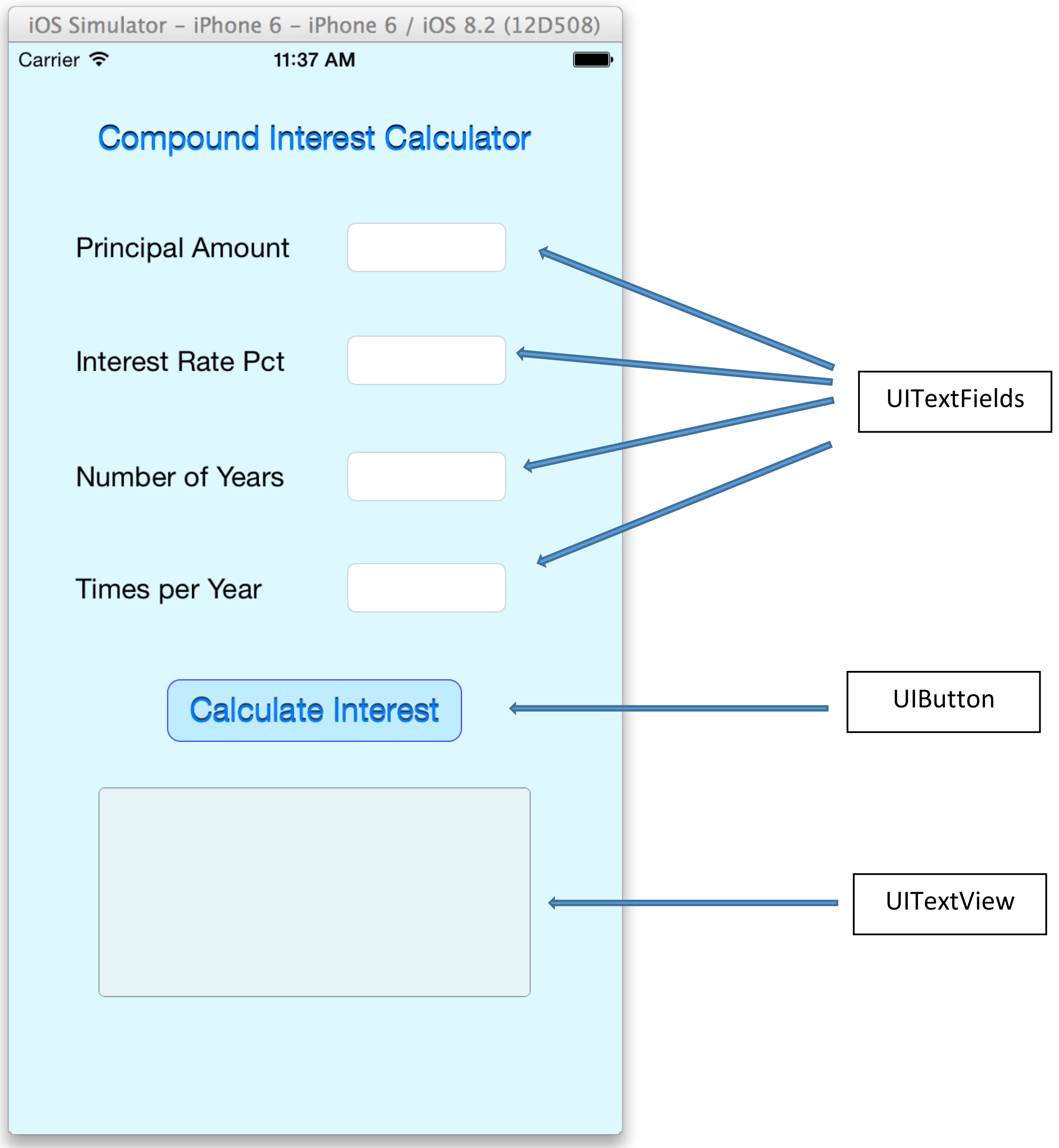
Tasks: (be sure to see ***Specs*** section below for guidance on required steps for performing these tasks)

Create a new Xcode Terminal App project using the following steps:

- Start Xcode (version 9 or latest) and choose *Create a new Xcode project*
- Under *iOS Application* in the left column, choose *Single View Application* and click ***Next***
- Fill out the following options for your new project
    - ✓ **Product Name**: enter *LP2_compoundInterestCalculator_yourUserName*
    - ✓ **Organization Name**: *assignment3 firstName lastName* (eg assignment2 Jon Cooley)
    - ✓ **Organization Identifier**: *edu.cvtc.Swift_yourFirstName*
    - ✓ **Language**: Make sure *Swift* is selected
- Click ***Next*** and *choose the folder where you wish to save your project*.
- Click ***Create*** and Xcode will open your project.

Your goal is to create the following iOS app project using Interface Builder (via **Main.storyboard**) to create the appropriate views and then code the *ViewController*, *Validator*, and *CompoundInterestHelper* classes to make the app functional as specified below.





The rest of the views are **UILabel** views that don't need to be modified.

1) Get input from the user, using ***views*** *(see above)*, for a principal amount (**P**), the annual interest rate (**r**) as a percentage (i.e user would enter 4.3 which would need to be used in your code as .043), the number of years (**t**) the amount is invested, and the number of times the interest is compounded per year (**n**).

   The formula you will use for this is: $\text{Result} = P(1 + r/n)^{nt}$

2) In your ViewController class declare the following instance variables and method:

Declare instance variables (connected as Outlets) for several of your views named:

- *principalTextField*
- *rateTextField*
- *numYearsTextField*
- *timesPerYearTextField*
- *resultsTextView*
- *calcInterestButton*

Create a single instance method (Action called when the button is tapped) named *calculateCompoundInterest*. No parameters will be passed to this method.

Declare the following instance variables in your ViewController class:

• *principalAmount*	*Double*	*(intial value: 0.0)*
• *interestRatePct*	*Double*	*(intial value: 0.0)*
• *resultValue*	*Double*	*(intial value: 0.0)*
• *numberOfYears*	*Int*	*(intial value: 0)*
• *timesPerYear*	*Int*	*(intial value: 0)*
• *showResult*	*Bool*	*(intial value: false)*
• *outputString*	*String*	*(intial value: "")*

3) In your ViewController class implement the following methods:

a. *viewDidLoad* (you will need to add some code to this automatically provided project method) as follows:

- Using a reference to your **button**, set the width and color of the border as well as rounding the corners to match the specs.

- Using a reference to your **UITextView**, set the width of the border as well as rounding the corners to match the specs.

b. *calculateCompoundInterest* which will be called when the "*Calculate Interest*" button is tapped. It should implement the following tasks:

- When (and only when) you have valid entered data in all your TextFields (you should use your *Validator* class methods to determine if your data is valid and have them re-enter any invalid data before going on), populate the TextView field with your results using a formatted string. It should look like this for example input data of a principal amount of 1500, an interest rate of 4.3, 6 for number of years, and 4 for number of times per year:

$1,500 invested at 4.3% for 6 years compounded 4 times per year is $1,938.84.

where each of the values should come from the input **or** your calculations. The amount value at the end of the string (1938.84 in our example) should be obtained by calling your *CompoundInterestHelper's* class method.

Use the NSNumberFormatter style, using the *currency style* to format the currency values you see above. Ask me to show an example of this in class.

- Refresh the user interface by calling the *refreshUI* method you create in the ViewController class.

c. *refreshUI* which should implement the following tasks:

- Reset the principal, interestRatePct, numberOfYears, and timesPerYear text field views to be empty and dismiss any keypads that are visible on them or the output text view by calling a *refreshUIKeyboards()* method you create in the ViewController class.

d. *refreshUIKeyboards* which should implement the following tasks:

- Use the *resignFirstResponder* method to dismiss any open keypads for the four text fields/views in our main view.

Custom Class Specs:

- Create a new class named **Validator** (subclass of NSObject) which will expose and define three methods:
  - **`isEmptyField`** which should accept one parameter in a parameter variable named *textValue* of type **String**.

    This method should check the passed-in String value returning ***true*** if the string is empty and ***false*** if it is not empty. Hint: there is a property of String objects that makes it easy to check if the string value is empty or not.
  - **`checkInteger`** which should accept one parameter in a parameter variable named *integerString* of type **String**.

    This method should attempt to *typecast* the passed in integerString value to **Int**.

If it successfully converted the passed in string to an Integer value, check the converted integer value returning ***the converted integer value*** if it is greater than zero. Note that if the typecast fails, it will return nil.

If the typecast failed with nil OR the converted integer value was not greater than zero, then return 0.

- **`checkDecimalNumber`** which should accept one parameter in a parameter variable named *numberString* of type **String**.

  This method should attempt to *typecast* the passed in numberString value to **Double**.

  If it successfully converted the passed in string to a double value, check the converted double value returning ***the converted double value*** if it is greater than zero. Note that if the typecast fails, it will return nil.

  If the typecast failed with nil OR the converted double value was not greater than zero, then return 0.0.

- Create a new class named **CompoundInterestHelper** (subclass of NSObject) which will expose and define the following method:

  - **compoundInterest()**

    This method will perform the conversions and calculations necessary to compute the amount at the end of the investment and return the result as a **Double**.

    This method will take in the following parameters:

principal	type **Double**
ratePct	type **Double**
numberOfYears	type **Int**
timesPerYear	type **Int**

    It should return a **Double**.

    The following tasks should be performed in this method:

    i. Break down the formula into pieces so we don’t end up with a very long, messy line of code

       1. Determine correct annual interest rate to use by dividing the passed in rate by 100.0 value storing the result in a variable named **interestRatePct**.

2. Divide the above calculated interest rate by the passed in number of times per year value (converted to a Double) storing the result in a variable named **divResult**.

3. Add 1 to the previous division result storing the result in a variable named **sumResult**.

4. Multiply the passed in *numberOfYears* by the passed-in *timesPerYear* and store result in a variable named **powerValue**.

5. Raise the previously calculated sumResult to a power of the just calculated powerValue and store in a variable named **amountToMultiplyBy**.

6. Calculate *and return* the amount result at the end of the investment by multiplying your passed in *principal* by the amount you just calculated by raising sumResult to a power of powerValue.

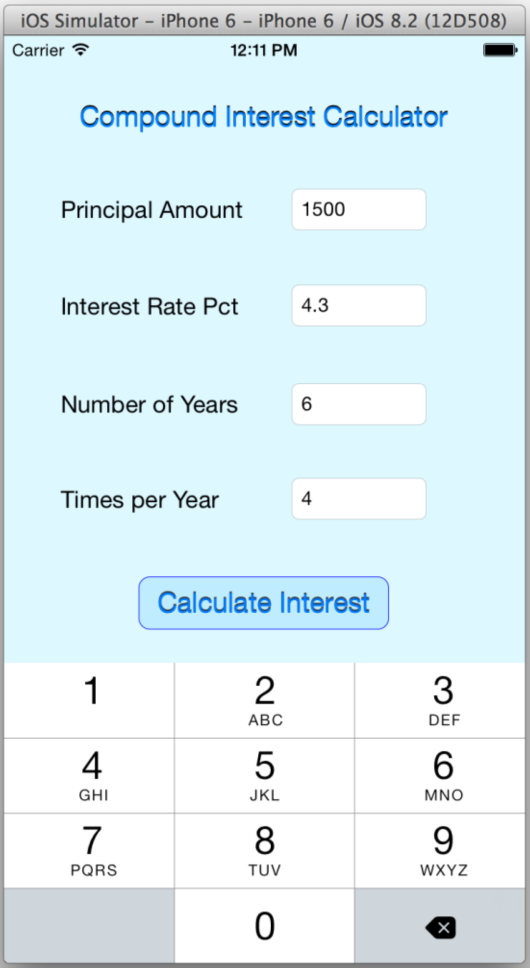
Any variables needed to store intermediate values should be declared inside the method so they are private to the method.

Deliverable:
Name your Xcode project **compoundInterest_yourNameHere** and when finished, zip up your Xcode project into a folder, and submit it in the provided Dropbox.

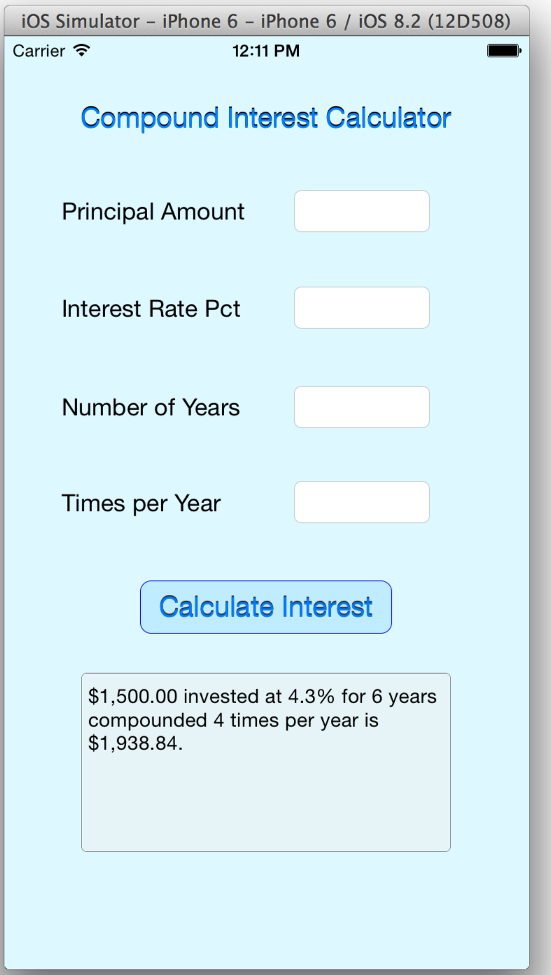
Remember to *closely follow specs*.

Screenshots

First two screenshots show valid results while next three screenshots show invalid data cases with no principal or an invalid principal amount being entered. Note that for the invalid principal examples, I am not showing any messages related to the other fields in these screenshots, but there would be additional messages for other invalid fields as well…

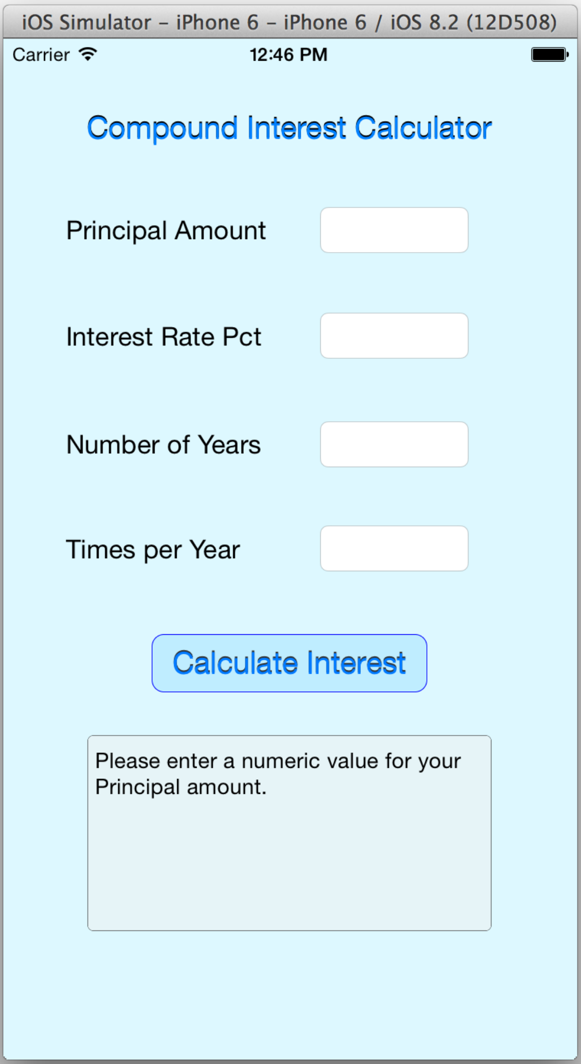


Data being entered

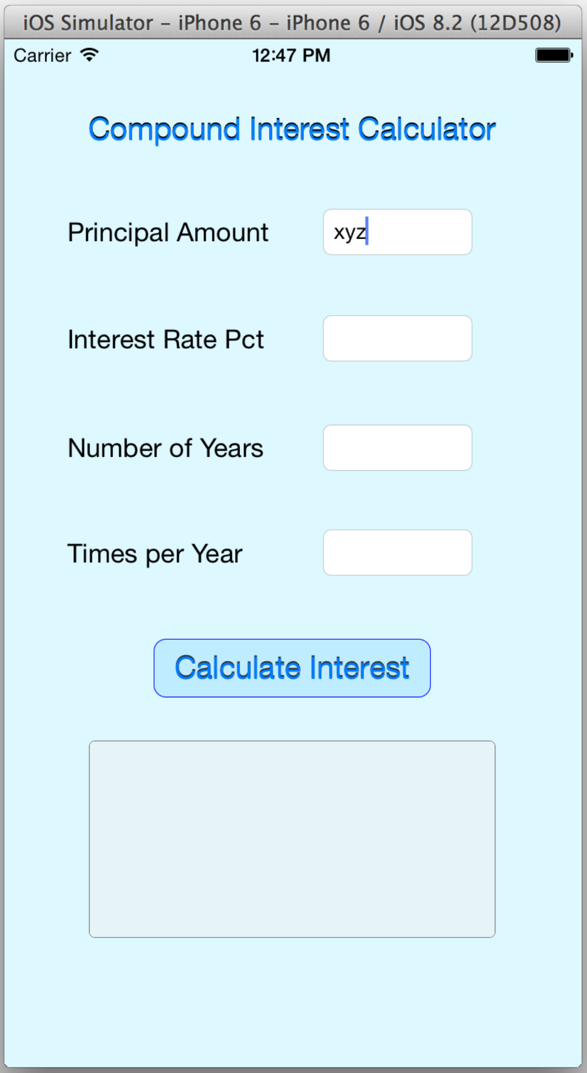


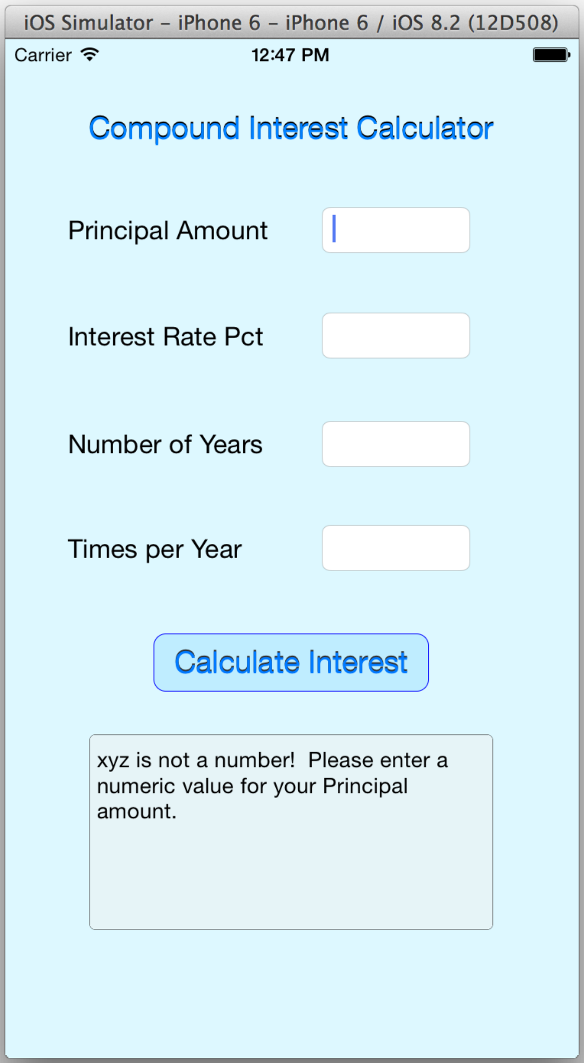
$1,500.00 invested at 4.3% for 6 years compounded 4 times per year is $1,938.84.

Valid result screen



Please enter a numeric value for your Principal amount.





xyz is not a number! Please enter a numeric value for your Principal amount.