Assignment 4: Graphing Calculator: Due October 13, 10:00 a.m.

Objective

We are going to continue to develop our calculator. This time we will create custom views, get more into the lifecycle of the application, and use a UINavigationController.

This assignment will build on last week's assignment. You'll be copying code out of it. Please note: you have two weeks to do this assignment. I gave you a little over a week to do the last assignment. I'll let you do the math there yourself, but you might want to start before the 12th.

Tasks

1. Create an application that when launched will present the use with your calculator, but this time inside a UINavigationController.
2. Your calculator should now only have one variable.
3. You need to add a button to your calculator, that when pressed will push a new UIViewController subclass onto the UINavigationController's stack, bringing p a custom view which is a graph. The graph will graphically display whatever expression is currently in the calculator. The x axis will supply the variable values while the y axis should display the results at those points. Pick a reasonable scale for your graph to start with.
4. In addition to the custom view, your graph should contain two buttons: one for "Zoom In" and one for "Zoom Out." These should change the scale accordingly.
5. Your code must display the axes of the graph. Code is provided to help you with this. The code will draw axes at a given origin and with a given scale.
6. Your graphing view must be generic and reusable. For example, your graphing class shouldn't know anything about CalculatorBrains or even expressions. Use a protocol to get the graphing view's data. Remember views should not own their data.
7. Make your user-interface nicer. For example, you can use colors to group buttons together visually. Make sure you provide titles for your UIViewControllers. Consider the user experience. E.g. if the user hits the "graph" button without having hit the "=" button you might want to supply it for them. The calculator UI will now matter.

Hints

1. When you create your project make sure you choose Window-Based Application instead of View-Based Application.
2. When you drag in your files from your old project make sure you check the box that copies them.

3. Once you have dragged in your old files, go into the application delegate's applicationDidFinishLaunching: method and add a UINavigationController, push your CalculatorViewController onto it and the call addSubview: as in the Psychologist example app. Then run it. Your calculator should show up good to go. If you have buttons that are cut off or missing you might need to revisit your layout. It is also a good time to learn about autosizing techniques in the Inspector.
4. When you make your new UIViewController to display the graph don't forget to click the "With XIB for user interface" button. That will allow you to add your buttons interactively.
5. This new view controller is just another MVC controller. Consider what its model ought to be. Of course it needs outlets into its View as well.
6. This will leave you with 3 view controllers. The one from your previous assignment, the navigation controller, and the new graph view controller.
7. Once you have created your new view controller (and its xib file) it is a good idea to go right in and wire up its buttons. Don't forget that you will be adding a generic view to the xib and that you need to set the class to match the new view that you are creating.
8. You'll need to add a button to your old view controller that will push the new view controller onto the UINavigationController stack. This new button will replace the old Solve button.
9. It is ok to call you generic graphing view's delegate data provisioning method repeatedly inside a drawing loop. In fact that's the best way to do it.
10. In your custom drawRect: method you'll want to use the helper code to draw the axes. Make sure you use the same scaling approach as is used there. All you need to do to use the helper class is first set up your graphics context and then call the one method it contains.
11. Implementing drawRect: is simpler than it may seem. You need to iterate over every pixel across the width of your view and convert (via scale and origin) the horizontal position into the coordinate space of your graph. Then you ask your delegate to give you the appropriate y value for that x. At that point you need to convert that y value back to pixel space (basically the reverse of the earlier process). Then you can draw the next point.
12. The best solution is not to draw lines from point to point (this breaks down with zooming, or discontinuous functions). It is ok though. There are extra credit opportunities here.
13. Be sure to iterate over pixels not point. Otherwise your graph will be horrible on a high-res iPhone or on an iPad. You'll need contentScaleFactor to help out.
14. Zooming in and out should be just a matter of changing the scale you are working with.
15. The Happiness and Psychologist demos were done for very good reasons.
16. Test your application on various expressions.
17. Memory management still matters.

Evaluation

The same criteria applies from previous assignments.

Extra Credit

1. In the hints section I note that there are problems with drawing graphs as lines from point to point. Another approach is simply to draw dots at each coordinate you calculate. This would at least help with the discontinuous function problem. It is up to you to figure out how to do this.
2. If you do the previous extra credit you may notice that some functions (e.g. sin(x)) look better when using the "line to" strategy than the dot strategy (depending on the amount of zooming). Try adding a UISwitch to the interface to allow a user to choose the drawing mode.
3. Clean up the descriptionOfExpression: output. This is an exercise in anticipating all the possible expression combinations and also in using the NSString class. So, for example, the expression x sin might be converted to sin(x).
4. See if you can get your view controllers to rotate properly . Shouldn't be too tough for the graph view, but it will be really hard for the keypad view. Lots of extra credit though! You will need to work on the springs and struts using the inspector on your xib files. And you'll need to make sure shouldAutorotateToInterfaceOrientation: returns YES more than it does by default.