Divide objects in your program into 3 “camps.”
Model = **What** your application is (but not **how** it is displayed)
Controller = How your Model is presented to the user (UI logic)
View = Your Controller’s minions
It's all about managing communication between camps
Controllers can always talk directly to their Model.
Controllers can also talk directly to their View.
The Model and View should never speak to each other.
Can the View speak to its Controller?
Sort of. Communication is “blind” and structured.
The *Controller* can drop a *target* on itself.
Then hand out an *action* to the View.
The View sends the action when things happen in the UI.
Sometimes the View needs to synchronize with the Controller.
The **Controller** sets itself as the **View**'s delegate.
The delegate is set via a protocol (i.e. it’s “blind” to class).
Views do not own the data they display.
So, if needed, they have a protocol to acquire it.
Controllers are almost always that data source (not Model!).
Controllers interpret/format Model information for the View.
Can the Model talk directly to the Controller?
No. The **Model** is (should be) UI independent.
So what if the Model has information to update or something?
It uses a “radio station”-like broadcast mechanism.
Controllers (or other Model) “tune in” to interesting stuff.
A View might “tune in,” but probably not to a Model’s “station.”
Now combine MVC groups to make complicated programs ...
MVCs working together
MVCs not working together
MVCs working together

What happens when your application gets more features?

Now all of your UI can’t fit in one MVC’s view.
MVCs working together

What happens when your application gets more features?

We never have an MVC’s view span across screens. So we’ll have to create a new MVC for these new features.
MVCs working together

So how do we switch the screen to show this other MVC?
MVCs working together

We use a “controller of controllers” to do that. For example, a **UINavigationController**.
MVCs working together

The UINavigationController is a Controller whose View looks like this.
MVCs working together

But it's special because we can set its rootViewController outlet to another MVC ...
MVCs working together

... and it will embed that MVC's View inside its own View.
MVCs working together

Then a UI element in this View (e.g. a UIButton) can segue to the other MVC and its View will now appear in the UINavigationController.
MVCs working together
MVCs working together

Notice this Back button automatically appears.
MVCs working together

When we click it, we'll go back to the first MVC.
MVCs working together
Calculator

CalculatorViewController

CalculatorBrain

digitPressed:
display

CalculatorGraphViewController

GraphViewDataSource

graphView
Calculator

CalculatorViewController

CalculatorBrain

digitPressed?
display

CalculatorGraphViewController

GraphViewDataSource

Add to Favorites
CalculatorViewController

CalculatorBrain

digitPressed:
display

GraphViewDataSource

CalculatorGraphViewController

?
When someone clicks in this table, we want to update the graph. How can we do that?
We CANNOT directly ask this Graph Controller to do it because we are (indirectly) part of that Controller's View.
We do it in the normal way a View can talk back to its Controller: delegation.