Intro to Object-Oriented Programming: Part 1

LAB 11X: Color Bars!

What are we going to do?

We're going to make a brief foray into the world of Object-Oriented Programming.

I will be a little unsettling… a little foreign. I ask that you not freak out. Take it as it comes.

As we get deeper into this OO suff, we'll end up with more vocabulary and context with which to discuss the topic.

Only through using it, can we begin to discuss and learn more about it.

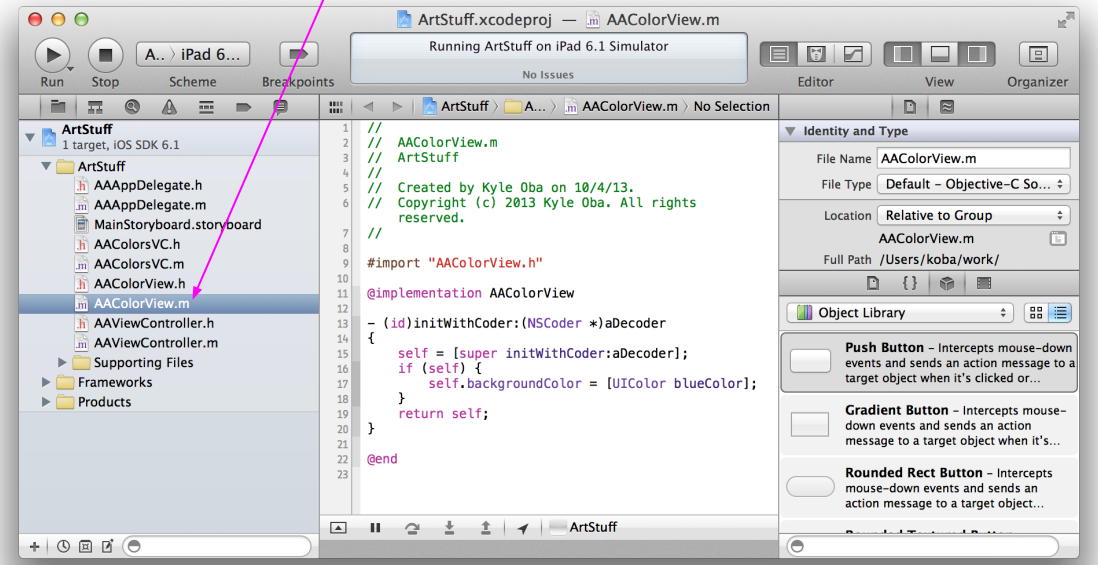
That said… let's build some color bars.

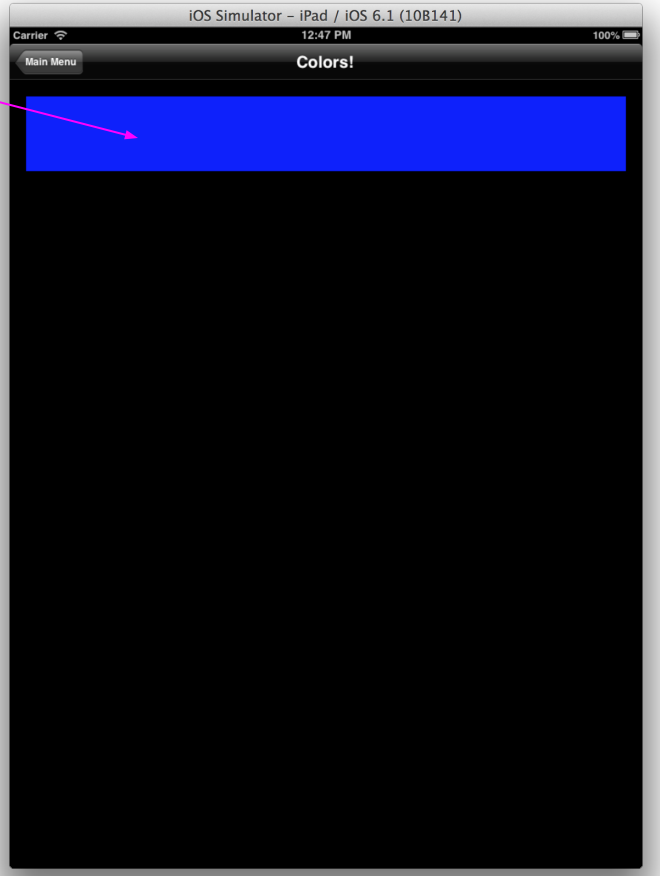
Get situated.

Open the **AAColorView.m** file.

Run it in the simulator. See how the bar turns blue?

Now, we're going to have it change color over time.





Classes: a closer look.

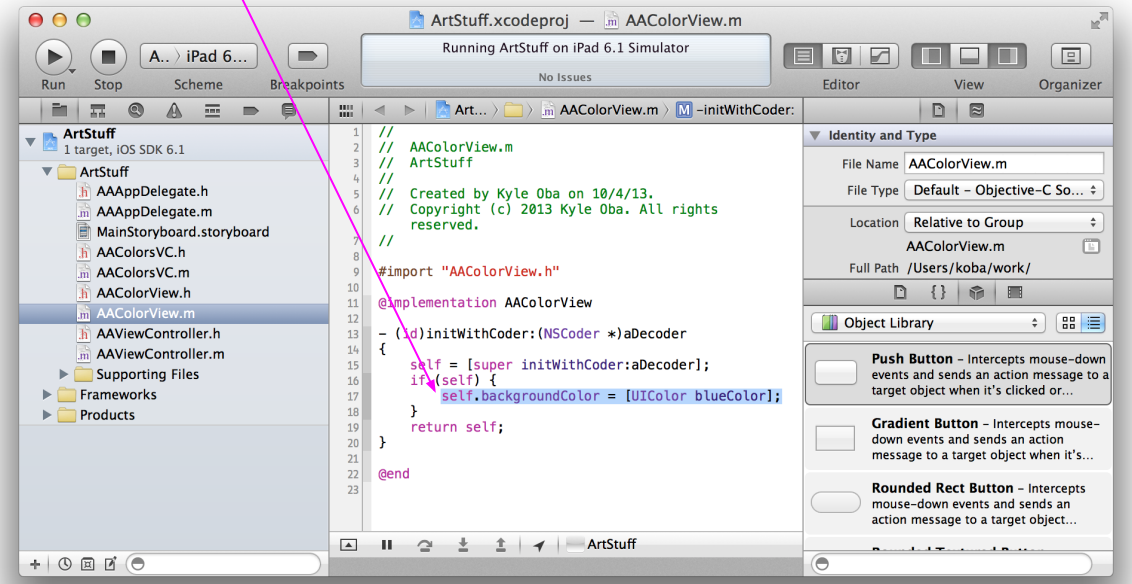
If you take a closer look at the **AAColorView** class, you'll see that we don't have a lot of custom code in there.

BTW: Custom code == stuff we're adding that's special.

It's really just this one line here.

What's happening here? When any **AAColorView** object is created, it's background color is immediately set to blue.

When are these created?



Storyboards and rehydration.

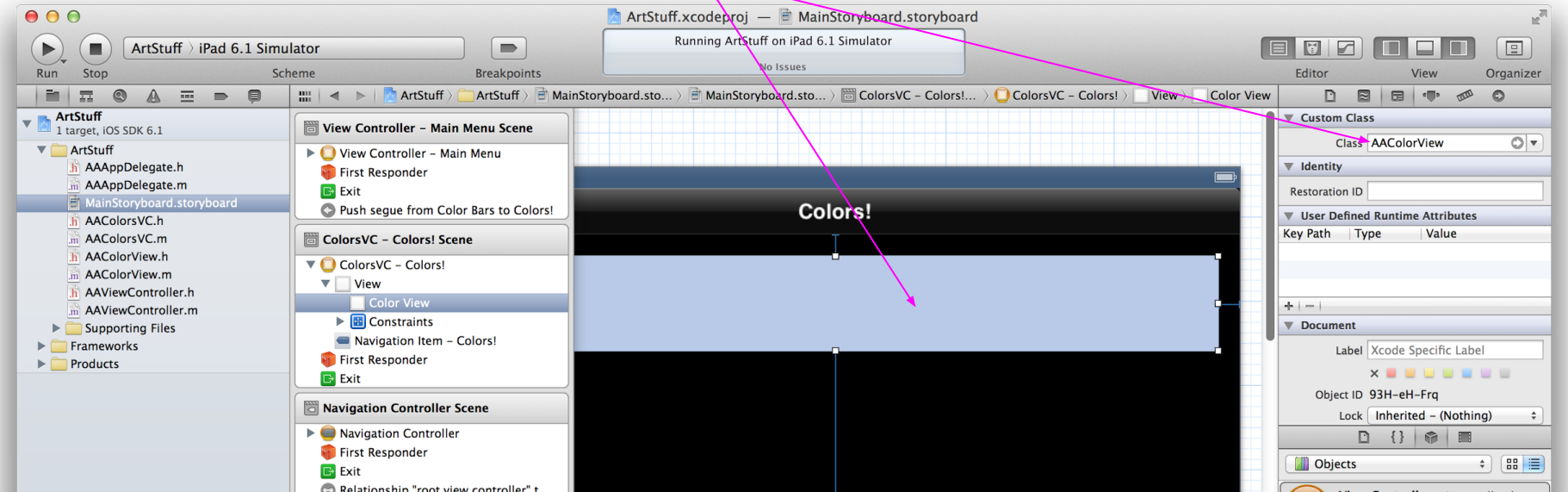
A Storyboard represents the screens in your app, but in a freeze-dried state.

When your app needs to display one of the screens, it must rehydrate the screen's objects.

How does it know how to rehydrate an object? It asks the object's class.

That white bar we added as a UIView, I selected it here. Remember we assigned it a Custom Class of **AAColorView**?

That's how that white bar knows that it is really an instance of the **AAColorView** class. So, it itself blue when it's created.



What is a AAColorView for?

Okay. So, what are we going to do with a **AAColorView** object? That white rectangle (that turns blue)? That **UIView** that we dragged onto the Storyboard?

Ideally, we'd like the color of the rectangle to change as time passes. We want it to display a color based on how much time has passed, kind of like a timer that uses color (instead of numbers) to show you how much time has passed.

To start, let's keep it simple

We are going to add code to **AAColorView** so that when you tell it a percentage a number (from 0.0 to 1.0), it will change color.

How do you change color based on a percentage?

First, let's look at color systems.

HSV Color

HSV stands for Hue Saturation Value. These are the three components of the color wheel at right.

Consider the top surface of this cylinder. Then, imagine that the far right-hand part of it is where 0 degrees starts from. You measure 360 degrees all the way around, going in the counter-clockwise direction.

If you look at the far left-hand portion of the wheel, you'll notice that halfway around the circle, the color is cyan (a light blue). That's where 50% is.

To calculate your percentage, you use an equation like this:

degrees / 360 degrees = percentage

so:

180 degrees / 360 degrees = 50%

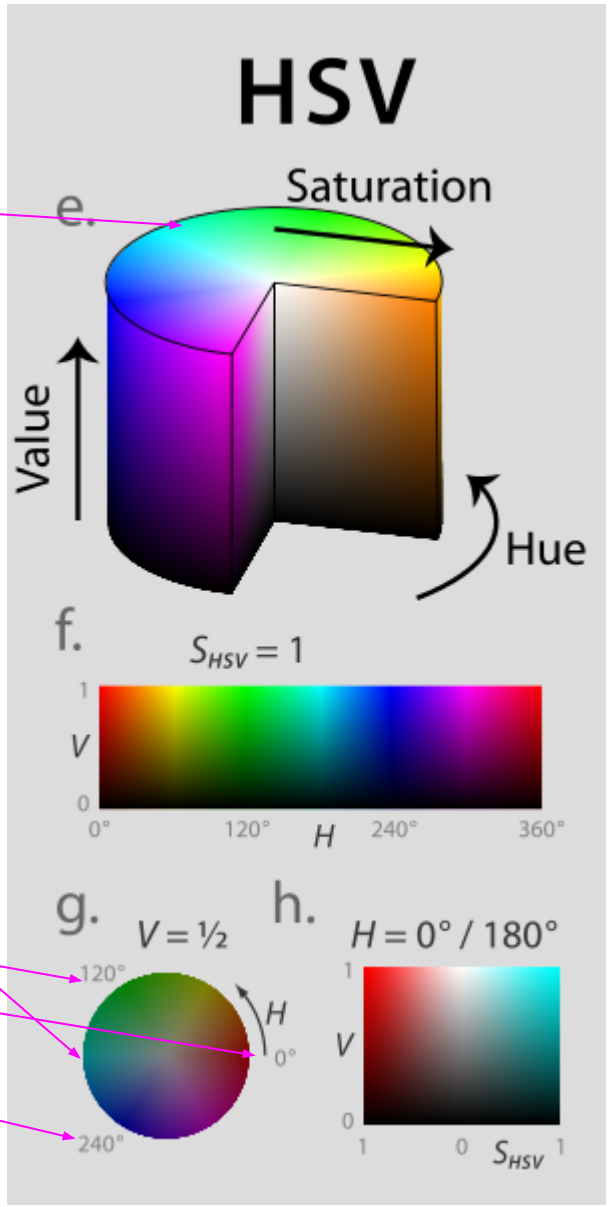
50% is where 180 degrees is on the circle. That's where cyan is. So 50% is a way of specifying the color cyan.

Note, 0 degrees (0%) is red, 120 degrees (33%) is green, and 240 degrees (66%) is blue.

So, we'll feed these percentage numbers to our **AAColorView** object and it will change its color according to how this wheel looks.

How do we do this?

Well, we need to add some code to **AAColorView**.



HSV image by Jacob Rus on Wikipedia, by CC BY-SA 3.0

This is just one color system

So HSV is just one of many color systems.

Another popular one is RGB (or, Red Green Blue), in which you specify relative amounts (from 0.0 to 1.0) of each component color.

I'm sure we'll see RGB soon.

The reason HSV is fun is that you can see *ALL THE HUES* by going from 0.0 to 1.0. With RGB you can't do that.

For now, let's get to it with HSV.

How do we change the background color using HSV?

Let’s change the background color.

Again. Change it again.

To do this, let’s get rid of the old code we used to change the background color to blue. I commented it out here (in green). You can just delete this line.

Now add the new **changeColorForPercentage:** method. This is a behavior. A verb. This is something that your new **AAColorView** object can do.

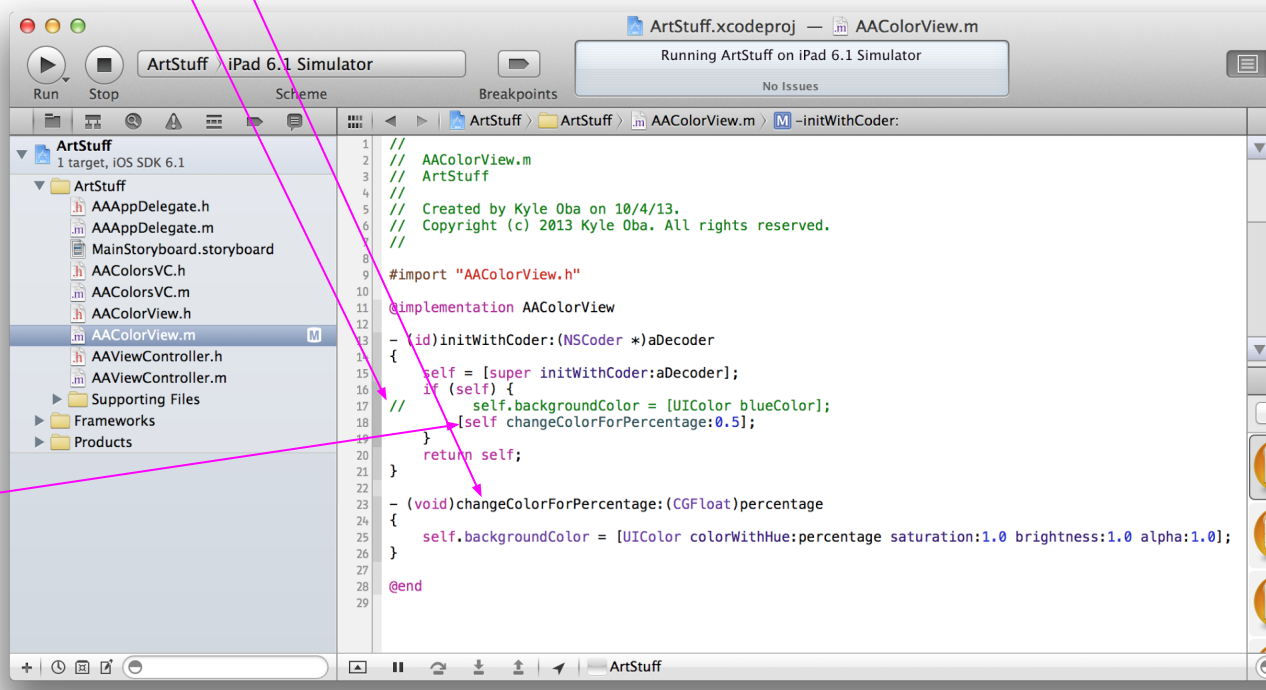
Take a look at that method. It takes the **percentage** parameter and feeds it into the **UIColor colorWithHue:saturation:brightness:alpha** method.

FYI, **brightness** is the same thing as **value** in HSV.

Also, this **alpha** thing is transparency. Fully opaque things have alpha of 1.0. Fully transparent things have alpha of 0.0. Things that are semi-transparent will have an alpha in the middle.

Make sense? If not, ask.

Finally, we call this method in the **init** block. The **init** block is ridonculous. Don’t worry about it too much. Just realize that this is for stuff that happens when the object is created.



Cyan!

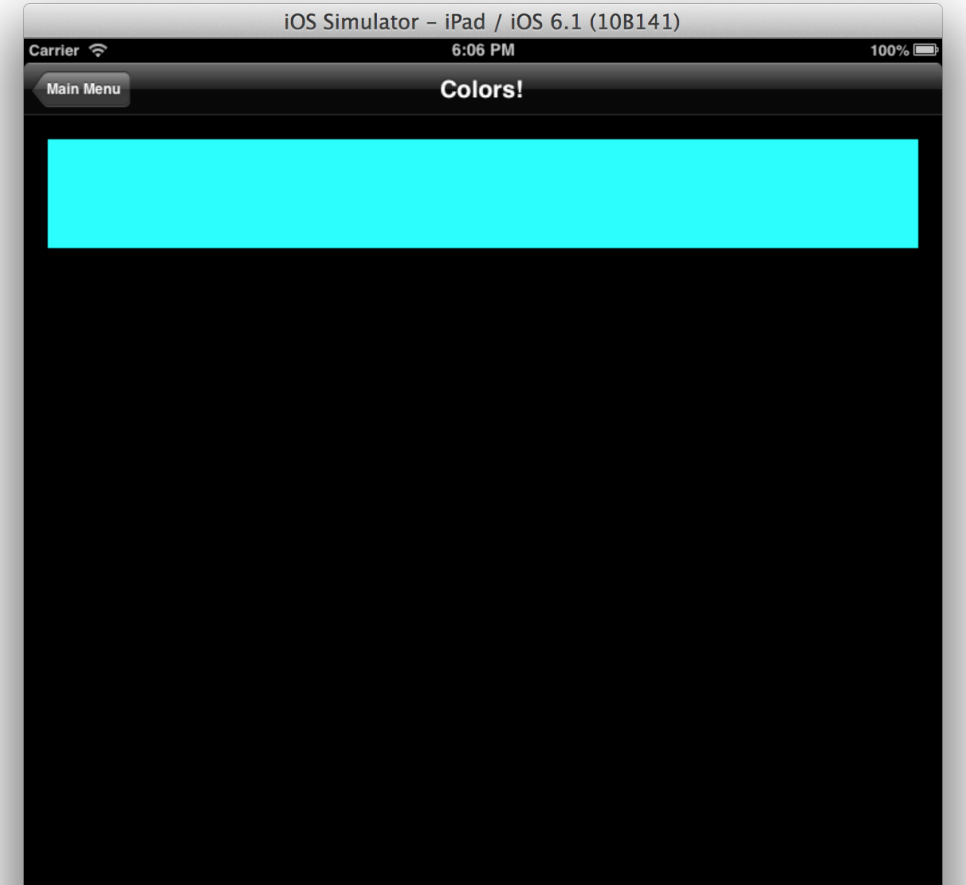
Run your app in the simulator and tap on over to the color bars screen.

And that's cyan! Cyan! CYAN!!!

That is exactly cyan. It's halfway between green and blue. Enjoy it.

Now, how do we make it change over time?

Timer? Sound familiar?



Another timer.

We had a timer back in our bouncy ball app.

We created it way back in LAB 06. Remember back then? Those days?

Go back to LAB 06 and review slides #26 to #32.

Do that all of that, but put the **displayLink** in the **AAViewController.m** file.

Do **NOT** put it in the **AAColorView** file. Just don't.

After you finish adding all the **displayLink** stuff, run your app. It should still work, but nothing should be visibly different.

A different approach to the timer.

We're going to change up the timer code in the **AAColorsVC.m** file here a little.

Take a look at my version here. It's not the same as what you have.

Make yours look like mine.

Here's what I changed.

Import the **QuartzCore** header file here.

Add a **displayLink** property here.

Instead of initializing your **displayLink** property in **viewDidLoad**, create a new **viewWillAppear:** method here.

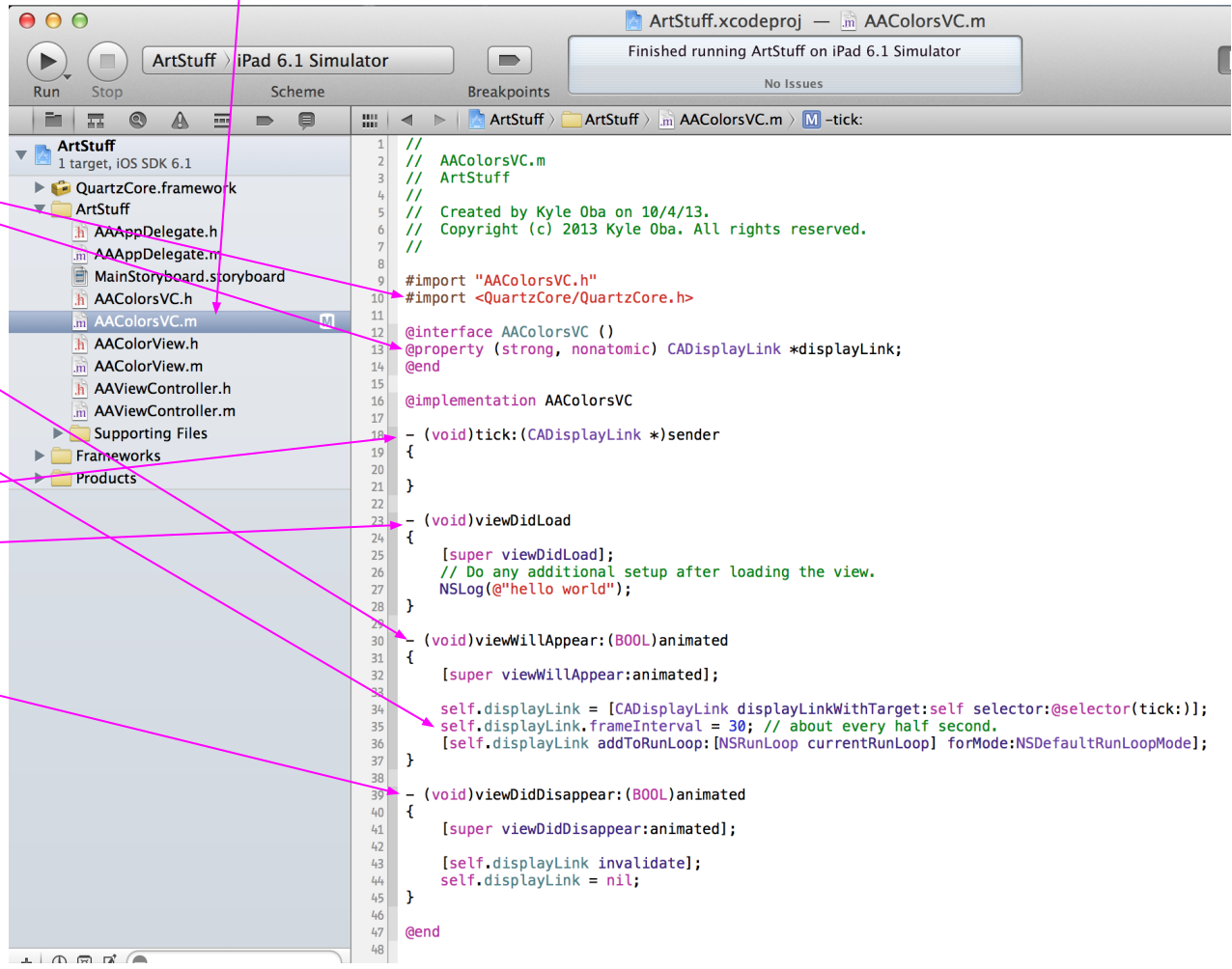
I added a new **frameInterval**. This is a clock-type thingy. So, let's fresh every half second just to be sure it looks clockish. (There are 60 frames per sec.)

Add your empty **tick:** method here.

Your **viewDidLoad** should be back to normal.

Add a new **viewDidDisappear:** method. This is used to clean up the timer. We have to do this now because your color bars screen is part of a larger app. It can come and go (when you go back to the main menu). When it "goes" you have to clean up after it.

That's a lot of code. We shall review it in class.



Run it again Sam.

Run your app. It should still work, but nothing should be visibly different.

Now what? We need to know what time it is. We need the time so we can figure out what the “seconds” value is of the current time.

The second hand of a clock runs from 0 to 59 and then repeats.

We’ll map this 0-59 value to a 0.0 to 1.0 percentage. See where this is going?

We’ll pass this percentage value to the **AAColorView**’s **changeColorForPercentage:** method. That will change the color of your rectangle ever 30 frames yo.

Okay...

Get at the seconds.

This is also a little crazy. We're need to know what time it is. But we only care about the *seconds* part of the time.

Like if it's 12:34 pm and 24 seconds. We only care about the 24 seconds.

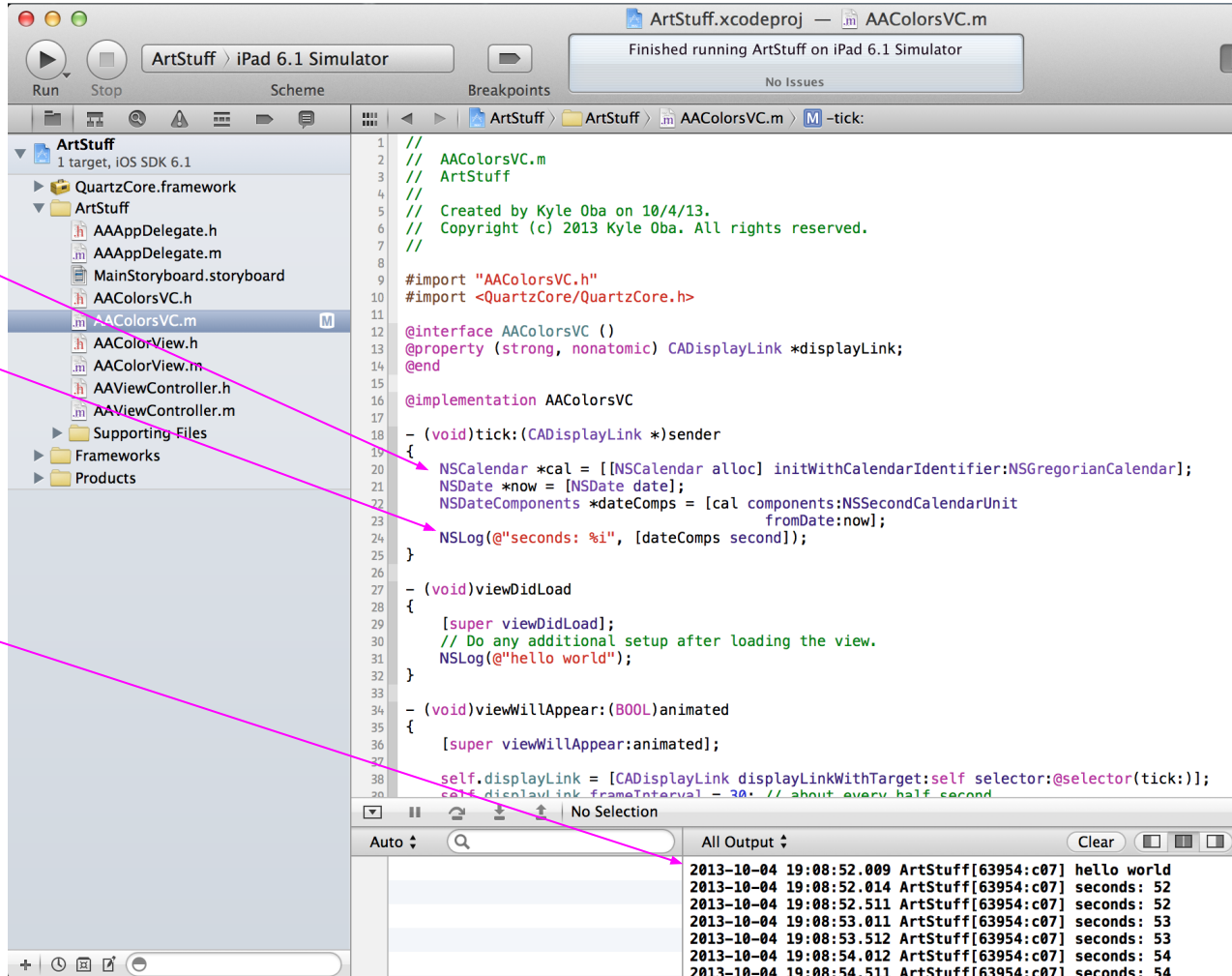
It's crazy and I had to look this up on the Interweb. But, this is how you do that.

Write this stuff into the **tick:** method.

See the **NSLog** function call? That's where we'll write the current seconds value to the log.

Run your app and go to the **Colors!** screen. You should see output like this.

See the seconds ticking by?



Connect the AAColorView.

We need to go back to the Storyboard to connect the **AAColorView** to the **AAColorsVC.m** file.

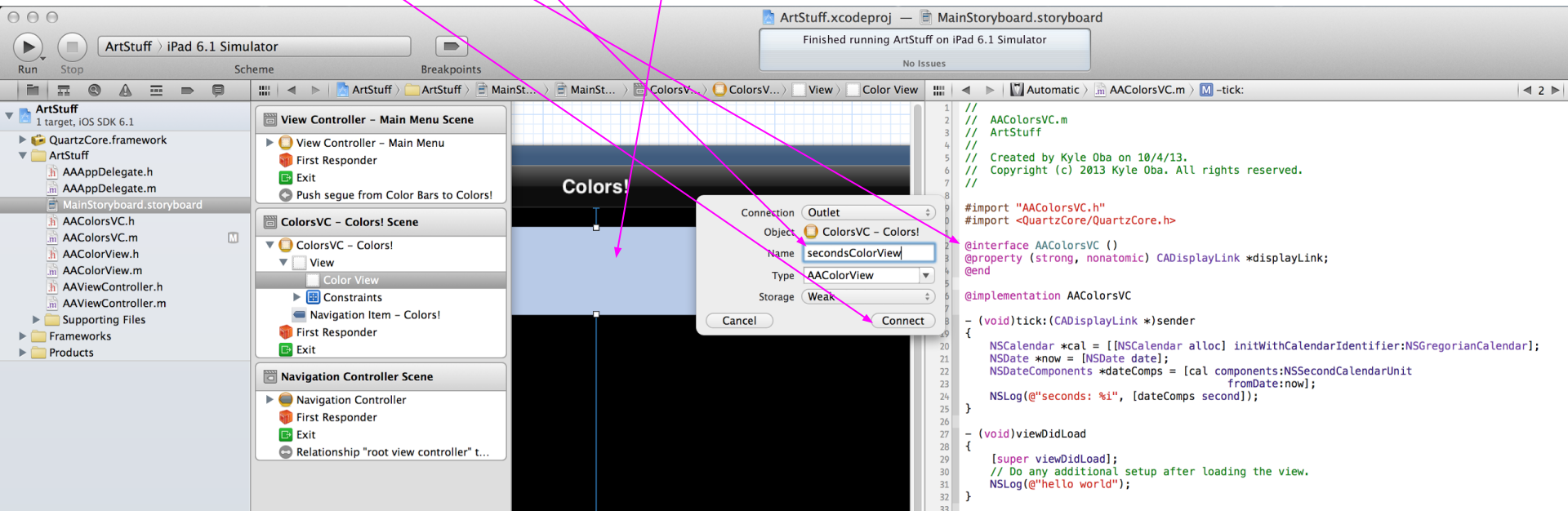
Why?

Because we need to be able to access it every time the tick: method runs. We need a handle.

We create a handle by creating and outlet. You've done this before. Hold down the control key and drag from the **AAColorView** to the **@interface** section.

Enter an outlet name. I used **secondsColorView** here.

Then press the **Connect** button.

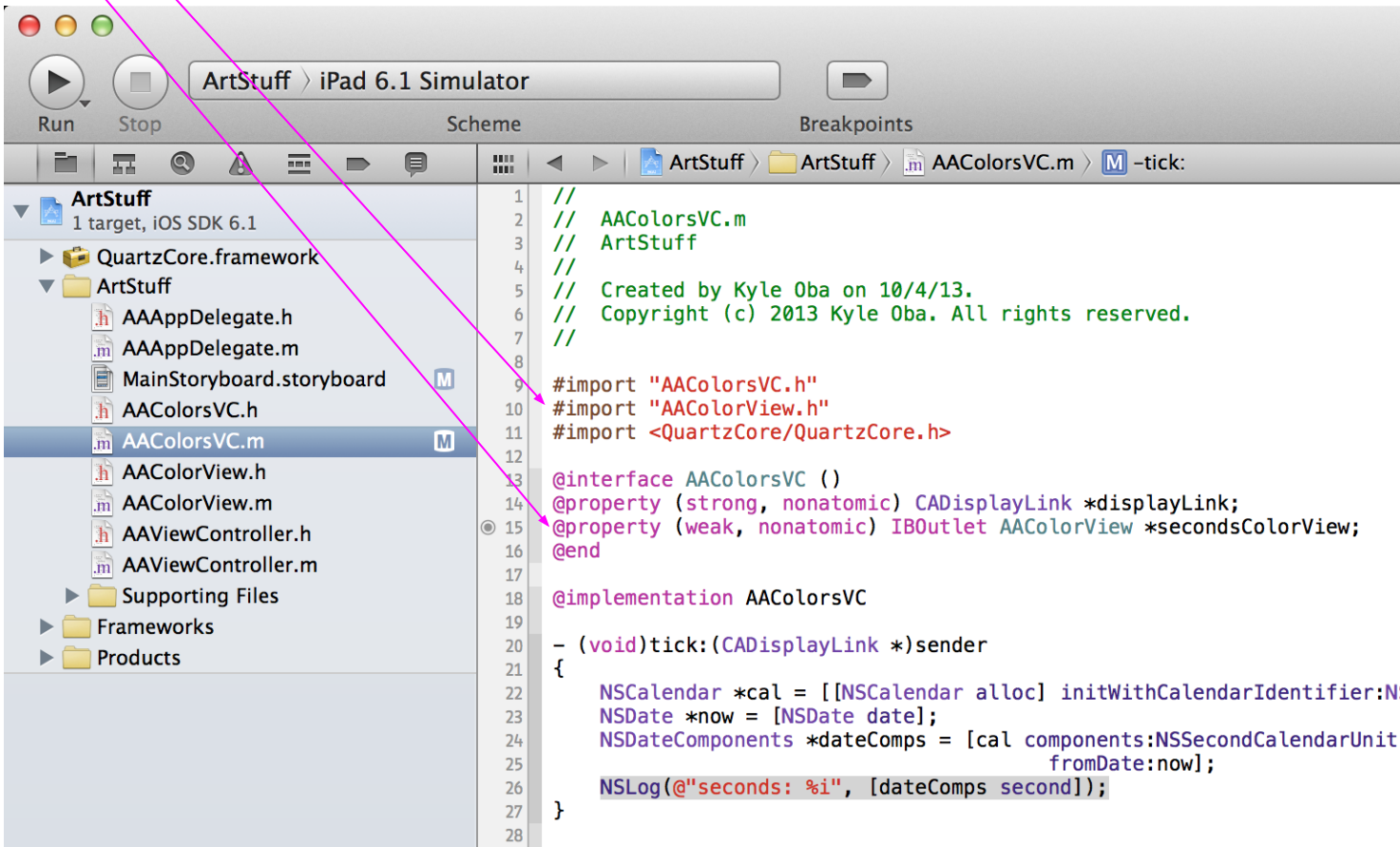


Add the property.

You'll see after you hit the Connect button that you've created a new **AAColorView** property.

But, you have one more thing to do here.

You must add the **AAColorView.h** header. Otherwise, Xcode will complain.



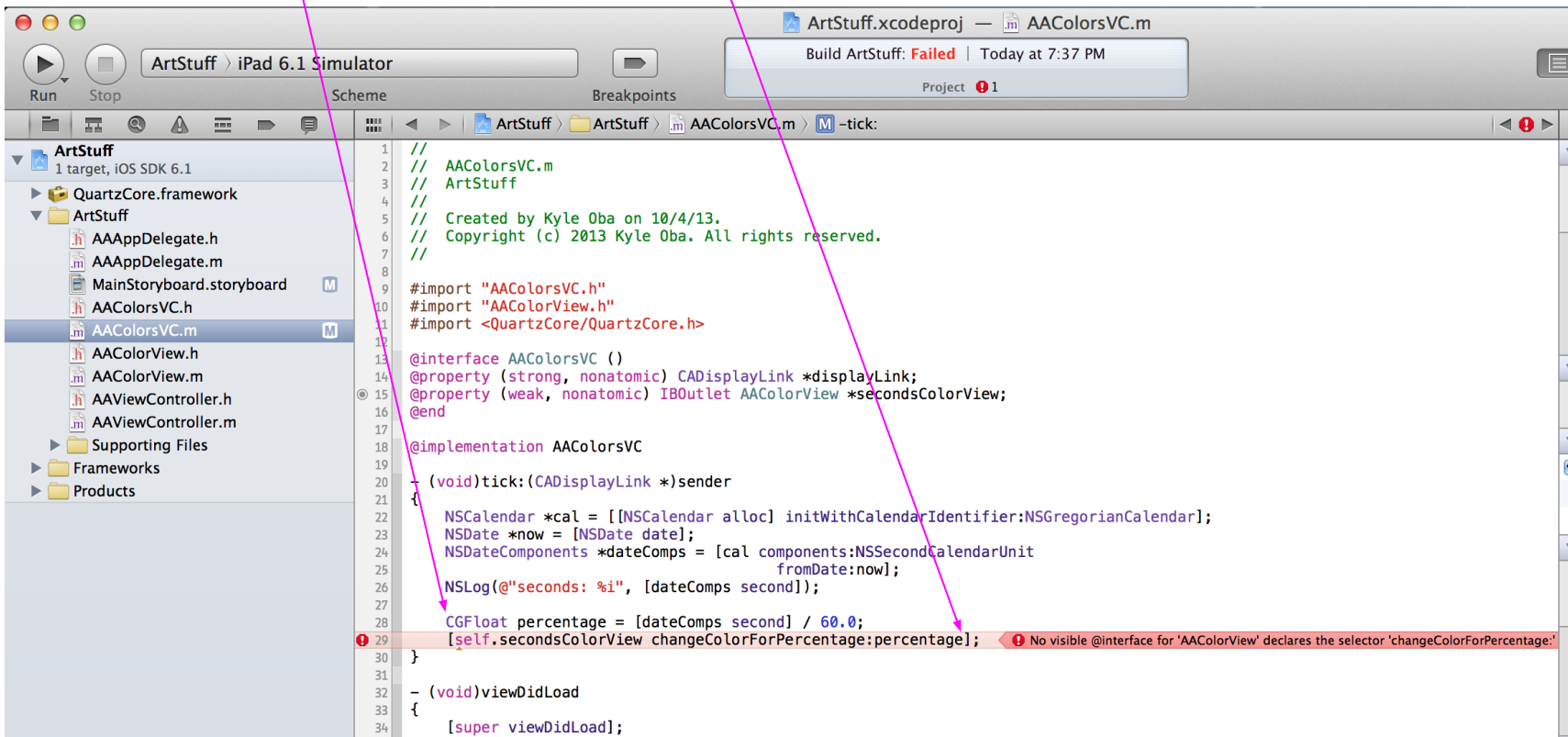
Percentages.

Now, let's calculate the **percentage**.

How do we do that? Take the seconds and divide by the number of seconds in a minute (HINT: There are 60 seconds in a minute.)

Now, call the **secondsColorView**'s **changeColorForPercentage:** method.

Try to run it. Failed!



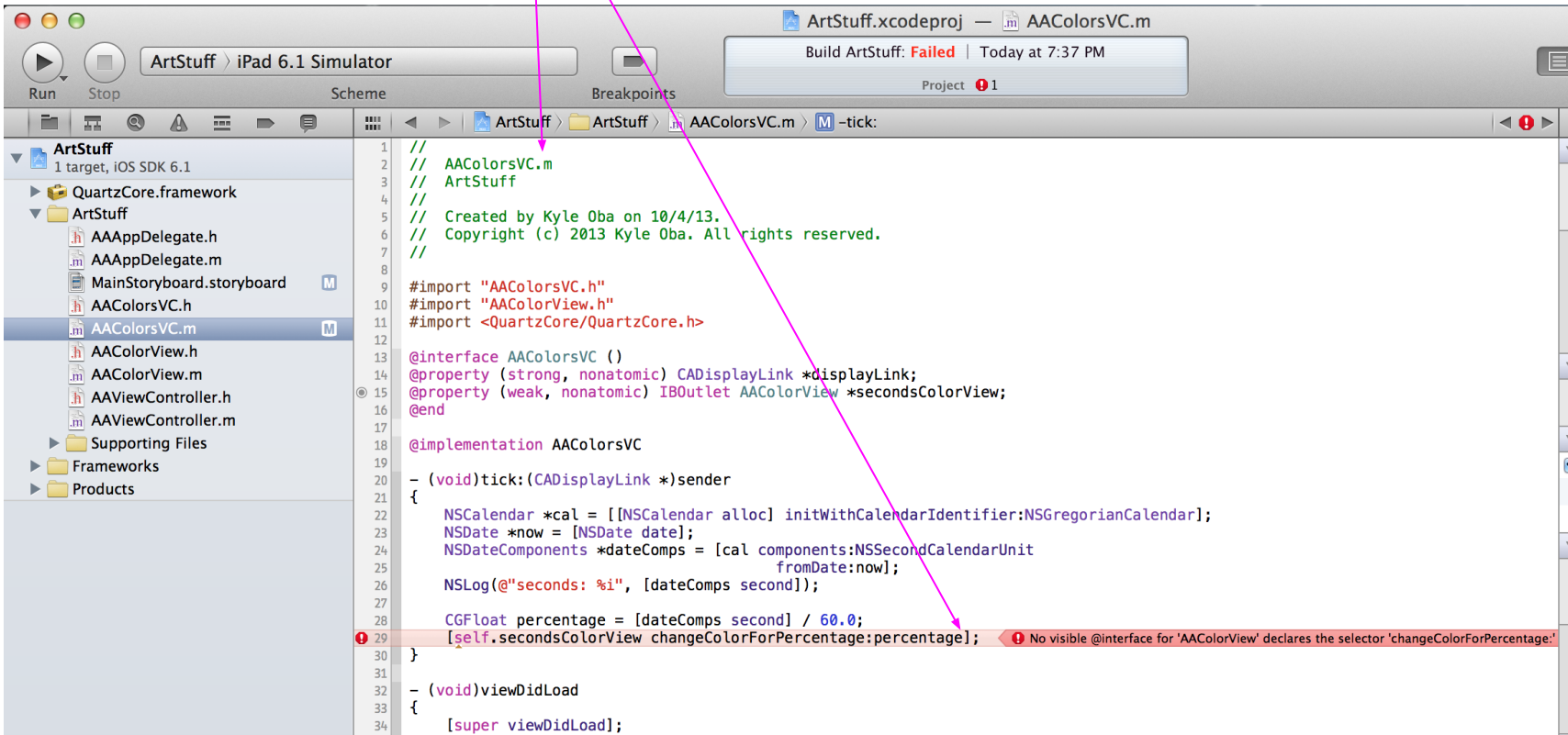
Why Failed?

Why did it fail? It failed because the **AAColorsVC.m** file doesn't know anything about the **changeColorForPercentage:** method.

Even though we're importing the **AAColorView.h** header file, we still have a problem.

The reason is that we didn't add the **changeColorForPercentage:** method. to the **AAColorView.h** file.

Let's do that now.



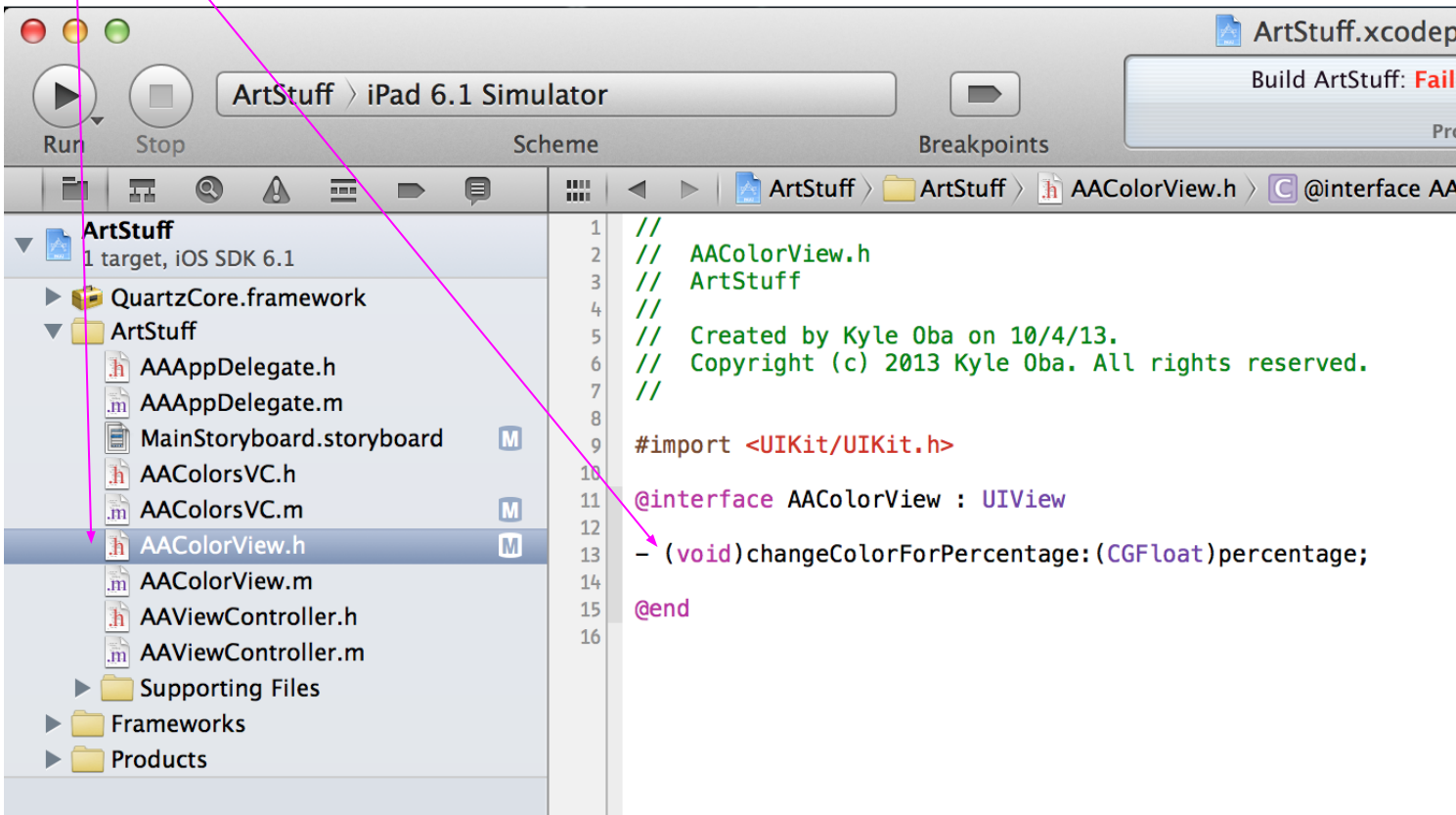
Update the header file.

Go to the **AAColorView.h** file.

It's empty.

This is what we use to let other files (classes) know about what methods **AAColorView** will respond to.

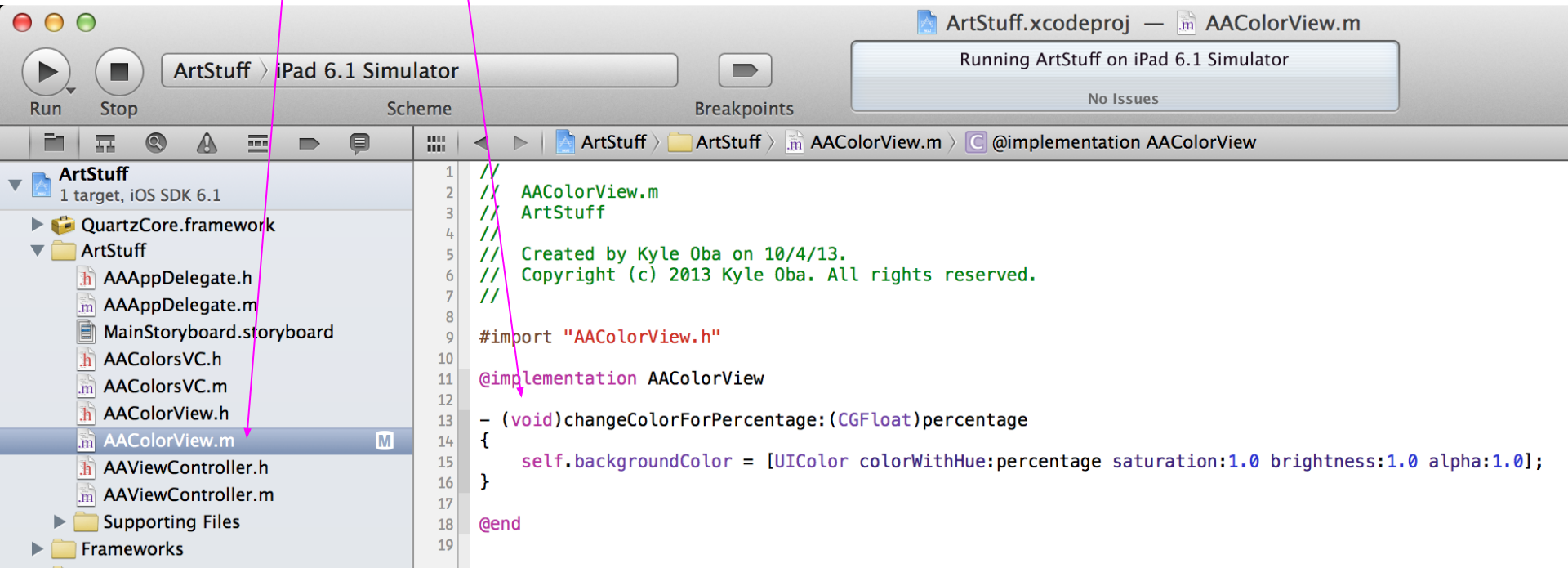
Add the **changeColorForPercentage:** method to the **@interface** section.



Clean up AAColorView.m

While we're at it, just remove the entire **initWithCoder:** method from the **AAColorView.m** file.

See, it's gone. It used to be here.



Look, look!

Now run your app. Watch the colors change. They should change about every second or so.

Once you compose yourself, show your app to someone you know. Explain to them that it’s a kind of clock. Explain how it works. Do a good job of this.

Seriously, that’s part of the assignment. Please do it.

What’s next?

We’re going to make two more bars. We’ll reuse our **AAColorView** class, so we don’t have to rewrite all that code.

Why do this? Minutes vs. seconds? Hours? Days?

How might we do this?

Conditionals?

Different data?