Lab Exercise 2

Objective: Create a simple application which demonstrates the various compositing modes.

Required files:

`CompositeView.m`	Implementation of the CompositeView class
`CompositeView.h`	Interface for the CompositeView class
`Solution/`	Directory containing one possible solution

Tasks:

You are given a partial implementation of the CompositeView class, a subclass of View to demonstrate the various compositing modes. The interface is defined in the header file, `CompositeView.h`, and the implementation is in `CompositeView.m`. The view defines three equal sized areas, draws into the first two, and creates the image in the third area by compositing the first two areas together using options specified through various methods.

Your task is to build an application around an instance of this class. The application should have a single window with the appropriate buttons and sliders to exercise all the features of CompositeView. Besides creating the interface, you also need to complete the implementation of the class; as it stands, it is not fully operational.

Do the following:

Run `CompositeLab` in the `Solution` directory and observe its behavior. This is one possible solution - yours might be better!

Create an interface for the program using the Interface Builder:

- First create an interface file for the application (use ªNew Applicationº from the ªFile'º menu) and save it as `CompositeLab.nib` in your `Lab2` directory.
- Teach Interface Builder about the CompositeView class: Open the Classes window and create a subclass of View. This should give you a class named Subclass1. Rename the created subclass as ªCompositeViewº and perform a ªParseº to read in the supplied header file. Double-click on the icon to verify that the parse went fine: You should see the actions of CompositeView in the inspector.
- Next define your interface (this is the most fun part!). Note that to create an instance of CompositeView you simply drag a CustomView from the Views palette and change its class through the inspector.
- Finally create a project for your application by using ``Project..." from the ``File" menu. You will need to add `CompositeView.m` as one of the class files.

Next implement the missing functionality in `CompositeView.m`. Note that you cannot use the class as it is; you will need to do some work first.

- Set the instance variables `dRect`, `sRect`, and `rRect`. Each rectangle should be the same height as the CompositeView and one-third as wide. (The `NXRect` structure is defined in the include files `/usr/include/appkit/graphics.h`; this file also contains some useful macros to access the fields of a rectangle. Or you can just use the function `NXSetRect()`.)

- Create the destination bitmap object and draw into it. (Hint: look at how it's done for the source bitmap.)
- Write the code for the methods that set the alpha-channel values for the source and destination bitmaps. (Hint: they should look just like `setSourceGray:` and `setDestGray:`.)

If you have some time left, create more complicated images in the source region. The code to draw the source bitmap contains a switch statement you can add to. See the `#define`s at the top of `CompositeView.m` for a list of suggested pictures. Allow the user to select the source picture through buttons in the window.

You will notice that `setOperator:` calls `display` to show the results of the new transfer mode. However, you do not need to redraw the entire view every time the transfer mode changes - all you have to redraw is the result box. Implement a method called `speedyDraw` which does the minimum amount of necessary work.