CSE 438: Mobile Application Development

# Lab 3: Drawing App

## Overview

In this lab, you will be exploring UIKit and Core Graphics through the creation of a simple drawing app. Additionally, this lab will require you to work with existing code, using object-oriented concepts to complete the functionality.

Please **read this entire PDF** prior to starting work on your lab. Every section contains information relevant to how you will be graded, not just the Requirements section.

There are a **lot** of online resources for drawing apps in Swift, but remember that you may not use a tutorial that specifically addresses the purpose of the lab, as this will be considered an academic integrity violation. You are welcome to use resources for enhancements or creative features, but please cite these with a comment.

**Please test opening your zip file on another computer to make sure the files are included properly. TA's are not liable for broken submissions and the late penalty will apply.**

## Details

**Due date:** Friday, October 8th, 11:59 PM CT
**Grading:** This lab is out of 100 points total. The exact point distribution is described in the "Requirements" section below.
Submission: Zip the entire project folder and submit it on Canvas. Please name the Xcode project **"FirstNameLastName-Lab3".** *Describe* your creative feature as outlined in the "Creative Portions" section in a readme.txt file in your folder.

## Description

This lab requires you to work with existing code and modify it to meet the requirements. Please download the files for Lab 3 from Canvas, and add them to your project. Ensure when you do that you check "Copy Files if Needed", or the TA will be unable to grade your lab.

The user should be able to draw shapes with different colors, move them, resize them, and rotate them. Additionally, they should be able to erase a shape and to clear all shapes on the screen. Shapes should always update live as the user move their finger, and they should update appropriately.

To gain a better understanding of the app's required functionality, please watch the demo in lecture or video on Canvas. If you need a requirement clarified, please ask on Piazza or in office hours before submitting–we are hoping to avoid regrade requests along the lines of "I didn't realize this was a requirement."

## Provided Code

The files provided are:

- **DrawingItem.swift** - Defines a protocol for a "drawing item," which is anything that can be shown on the screen. A drawing item needs an initializer, a "draw" method, and "contains(point:)" which returns true if a point is within the drawing item. YOU MAY NOT MODIFY THIS FILE.
- **Shape.swift** - Defines a class, Shape, which implements the DrawingItem protocol. You should fill in the implementation of this class. However, you are free to do so in any way you wish. Remember that a shape must keep track of location, color, size, and rotation. Additionally, your shape should not be constantly re-calculating their bezier paths, but rather should store the bezier path, only updating it when needed.
- **DrawingView.swift** - Defines a UIView subclass that serves as the canvas. You should create a view in storyboard, and using the Identity Inspector, set the class of the view to DrawingView. This class draws an array of DrawingItem objects to the screen, automatically redrawing whenever the array changes. However, you may have to call setNeedsDisplay other times. YOU MAY NOT MODIFY THIS FILE.

Other than these, you are free to implement the project however you wish. You must use these three files as described. For more guidance, see the Helpful Advice and Code Snippets section at the bottom.

## Grading

**[20 points]** Users can draw three or more different solid shapes by choosing a shape and tapping the screen.
**[10 points]** Users can draw with five or more different colors.
**[10 points]** Users can move shapes around in "move" mode after drawing.
**[10 points]** Users can resize shapes using a pinch gesture.
**[10 points]** Users can rotate shapes using a rotate gesture.
**[5 points]** Users can delete shapes in "delete" mode by tapping on them".

**[5 points]** Users can clear all shapes on the screen.

**[10 points]** Code shows thoughtful OOP design and adheres to Swift guidelines.

**[20 points]** Creative portion: Add 2 other small features. Be creative!

## Creative Features

For this lab we want you to incorporate two creative features. A combination of many smaller features is feasible but it will make your description of them hard. If you do choose many smaller features try to make sure they can be grouped towards two greater goals. A “creative feature” doesn’t have to be a literal feature, it can involve an abstract concept. Try to defend your creative decision in your head, if you struggle to justify the significance then it’s probably not enough. A laundry list of small features would also be hard to defend. Put yourself in the shoes of an app developer — what is a significant improvement you can make to this app? Start brainstorming early!

Once again, in your readme.txt please defend your creative feature, use it as your chance to tell us why it is worth 10 or 20 points.

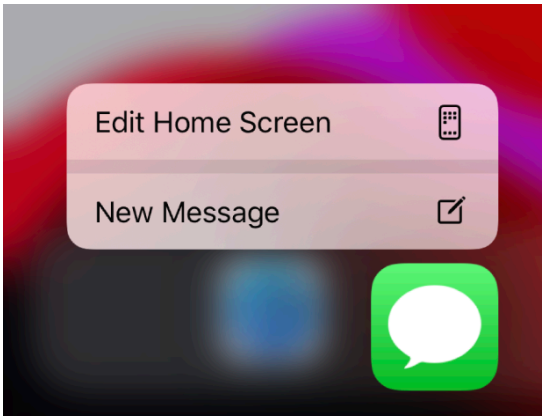
Example format:

1. **What you added (and how to use it)**
2. **How you added it and what the process involved**
    1. How does it compare to the effort required by the other portions?
3. **Why you added it**
    1. Why it is a meaningful improvement to the drawing app?

## Extra Credit

**[5 points]** Implement two subclasses of Shape, SolidShape and OutlineShape, each with a unique draw method. Using these classes, allow for drawing of three different shapes, either solid or outlined.

**[5 points]** Implement at least two Home Screen Actions for your app. See the image below for an example.




## Warnings

- You should not be recalculating every shape on screen every time a change is made.
- Every change should update live, not when the user releases their finger.
- Shape transformations should be smooth and work as expected.

## Helpful Advice and Code Snippets

**Using the DrawingView Class**
We've provided DrawingView, a subclass of UIView. To use it in Storyboard, you should add an empty view, open the Identity Inspector, and set the "class" field to DrawingView.

**Getting touches**
The first challenge will most likely be getting data from the user's input. The way to do this is by overriding the touchesBegan, touchesMoved, and touchesEnded functions in your view controller. After these are in place, use the following code to get the location of a user's touch.

```
guard touches.count == 1,
      let touchPoint = touches.first?.location(in: view)
else { return }
```

This function takes the first touch from a set of UITouch objects provided by the system, and finds its location in the current view. This limits the user to only inputting a single touch at a time, but it should suffice for the purposes of this lab. You will need this to draw, move, and delete shapes.

**Gesture Recognition**
However, gesture recognition (pinch and rotate) will likely be easier using UIGestureRecognizers. You can add a UIPinchGestureRecognizer and UIRotationGestureRecognizer to the view controller in Storyboard, and hook up the IBAction to the view controller class.

In these methods, consider using *drawingView.itemAtLocation* to get the item that these gestures are touching.

Additionally, you will run into an issue where the view controller won't fire both gesture recognizers at the same time. To fix this, declare your view controller as implementing UIGestureRecognizerDelegate, assign the view controller as the delegate to both your gesture recognizers, and add the following method:

```
func gestureRecognizer(_ gestureRecognizer: UIGestureRecognizer,
shouldRecognizeSimultaneouslyWith otherGestureRecognizer:
UIGestureRecognizer) -> Bool {
      true
}
```

Finally, you will likely run into an issue where resizing shapes does not work as expected. Note that UIPinchGestureRecognizer gives you a *scale* value that starts at 1 when the gesture begins, and will not be recalculated until the gesture ends. Thus, if you are multiplying *scale* * *current size* to get the new size, your size will likely grow/shrink exponentially and not look right. You need to keep track of what the size was when the gesture begins.

**Representing Different Shapes**
There are multiple ways you can keep track of the different shapes. Think about what is similar between them–how do they store color, size, location, etc? What differs between the different shapes?

You may want to make multiple subclasses of Shape (maybe Rectangle, Triangle, Circle, etc) for each different shape, and override some methods. Or you can use a property in Shape to keep track of what type of shape it is. Or there may be some other way–it's up to you!

**Input Mode**
There are many different input modes: drawing different shapes, moving, erasing. You may wish to keep track of these using an enum, and you may wish to use *enum associated values* to represent some of this information. Remember that you can earn additional points for exceptional OOP design.

**OOP**
Lastly, this application provides a great opportunity to leverage object-oriented programming principles to make your code easier to read and maintain, as well as save you time. If you are going to make a button for seven different colors, consider creating a UIButton subclass so you only have to write your button styling code once.

Think about structuring your app according to the model-view-controller paradigm. What is the model? What is the view? How do we connect those pieces in a reusable way? Contemplating these questions may seem trivial for a simple app, but becomes critical as your codebase scales.

Good luck and have fun!